vr_logic_lib documentation

This documentation was written by Vic for usage with v11 of the “vr_logic_lib”.

Welcome!

This library provides three tables that provide very powerful modding possibilities. Albeit it also gets complicated here. So yes, the power offered by this library comes at a price.

In this library we first find the “Logic Table” that allows doing a specific SET Logic call if a certain IF Logic call returns >0.

Note if no IF clause specified (zero length string) the SET always works. Like for example: changing the VP of a certain Hex. Or giving PP to a Regime on a certain condition.

Then secondly there is the “Change Global Libvar” table that allows you to change the variable value of a global libvar of any library. Like for example: when you want to change the last round for the scenario.

Thirdly there is also the “Flag Instructions” table that allows you to let the IF and SET logic in the “Logic Table” to access directly any stringlist. Like for example: when you want to change the fuel amount delivery.

The order of execution is:

- 1st the Logic Table
- 2nd the Change Global Libvar Table

Logic table

Both IF and SET columns allow scripting in a simple custom scripting language. In the table just below here is all the syntax possible.

abs(x)	Returns positive value. abs(-5) returns 5
inv(x,y)	Returns (100-x) * y / 100
rnd(x,y)	Random number between x and y (including x+y)
min(x,y)	Returns the lowest of the 2 numbers
max(x,y)	Returns the highest of the 2 numbers
dth(x,y)	Rolls xdy. dth(2,6) returns 2d6 roll.
chk(a,b,c,d,e,f) chk(a,b,c)	a=check function number b=variable 1, c=variable 2, etc..
exe(a,b,c,d,e,f) exe(a,b,c)	a=execute function number b=variable 1, c=variable 2, etc..
+	add
-	subtract
*	multiply
/	Divide. Keep in mind decimals are rounded.
>=	Bigger or equal return 1, otherwise 0
>	Bigger returns 1, otherwise 0
<=	Smaller or equal return 1, otherwise 0
<	Smaller returns 1, otherwise 0
==	Equal returns 1, otherwise 0
!=	Equals returns 0, otherwise 1
=	Sets a FLAG (INSTRUCTION) to another value.
|	Logical or... If left or right side is 1 then return 1, otherwise 0. For example 0|1 returns 1 and 0|0 returns 0.
&	Logical and.. If left and right side is 1 then return 1, otherwise 0. For example 0|1 returns 0 and 1|1 returns 1.

Multi line scripting

You can use ";" semi-colons to separate lines of instructions. In the case of IF Logic all lines must return >0 (true) for the total script to return >0 (true).

In the case of "SET Logic" all lines will be executed.

Variables

You can use the SET Logic to set a new variable to a value, like this: "MYVAR=100". Specifying a variable not yet created also will create it.

You can read a variable in IF Logic like this “MYVAR<100” or “MYVAR==100” for example.

Note the “=” for assigning a value and the “==” for checking a value.

If you are IF checking a variable not yet defined using SET it will have value 0.

Variables hardcoded

TURN = Returns 0 for first player (Regime Slot 0), and 1 for second player (Regime Slot 1)

ROUND = Returns the round number

Setting variables in the IF instruction

Also note that when using multi-line in “IF Logic” you can use the initial lines to give some SET Logic instructions. For example: “TEMP_VAR=dth(1,100); TEMP_VAR>50”. This multiline IF check first assigns a random value between 1-100 and then checks if it is higher than 50. Using temporary variables like this can help with complex calculations.

IF Logic elsewhere

Note there are many tables in different libraries that allow supplying IF Logic.

Change Global Libvar Table

This is very straightforward.

If the IF Logic returns >0 then in the specified Library name , the specified Global Libvar will be changed to the value specified in the last column.

Note that the value can not be set to a variable.

This is something that I might well add in the future.

Flag Instructions Table

This is maybe the most difficult to understand. Feel free to ignore this. It is really dense and often not necessary.

Here you can define a sort of command you can use in “IF Logic” and “SET Logic” to directly address a certain cell in any stringlist.

The table has 8 columns:

- Col 0 = The name you’ll use in the IF/SET Logic: For example “FUELDEL” or “GERINFREINF”
- Col 1 = Actual Stringlist ID that it refers to. This is the current ID, not the ID used in its library.
- Col 2 = Actual column # to look up flag Instruction Key’s value (or -1 for none)
- Col 3 = Flag Instruction VALUE that needs to be present in that column (specified in col2)
- Col 4 = Actual column # to look up flag instruction Key’s value (or -1 for none)
- Col 5 = Flag Instruction VALUE that needs to be present in that column (specified in col4)
- Col 6 = Actual column # for any behind the “.” variable name usage. (or -1, probably best for DC5 in almost all cases)
- Col 7 = Return column / Set Column # (depending on how the flag instruction is used)

So for example setting the German Fuel Supply differently in the Solsty scenario:

- Col 0 = GERFUEL
- Col1 = 86 (will depend on you scenario, look it up, using the “Fuel Table” in VR Basic Lib here.
- Col 2 = 1
- Col 3 = 0 (x coord)
- Col 4 = 2

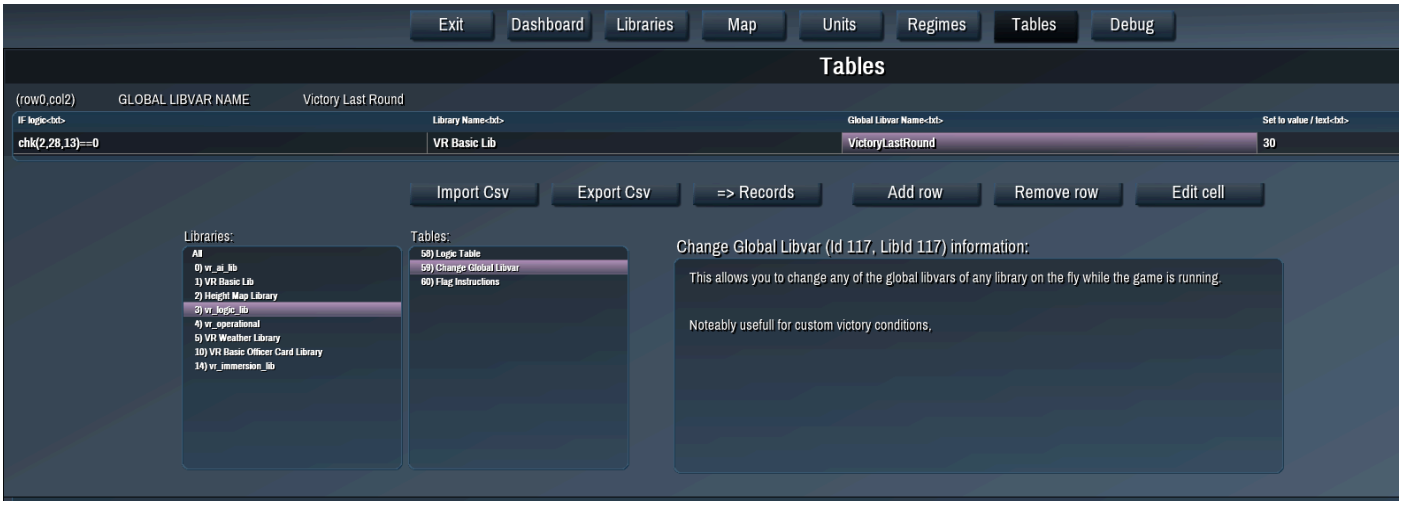
- Col 5 = 6 (y coord)
- Col 6 = -1
- Col 7 = 3

This example would result in GERFUEL being usable in SET Logic like GERFUEL = 5000 which would cause 5000 fuel per turn to arrive for the Germans.

Usage Examples

I'll start with one example.. But feel free to demand other specific examples in the forums or on Discord. If i find the time, I'll oblige.

If specific town conquered, the game will last longer



Note the Chk(2,X,Y) checks the owner regime slot #. 0=for germany, 1=for soviets. So here we say if this hex is owned by germany we adjust which round in this scenario is the last round.

DECISIVE CAMPAIGNS

Twilight in the East
Event scripting reference

Checks & Execs | Numeric IF / SET Logic edition

17 November 2026

A practical, numeric-only reference for DC5 scenario authors. Entries list the Event Editor argument names in call order, describe the numbers returned by Checks and explain what Execs change in a scenario.

172	199
CHECKS	EXECS

IF Logic and SET Logic: numbers only

DC5's IF Logic and SET Logic fields use numeric arguments for Chk(...) and Exe(...) calls. Do not pass words, quoted text, filenames, library names or other textual data. The commands listed below are the numeric-use subset of the scripting system.

The important distinction is the value, not the underlying program datatype: "25" represents a number, whereas "Berlin" represents text. Some functions store numbers in string fields or in StringLists; they remain useful here only when the supplied and returned contents are numeric. Commands requiring non-numeric content or returning descriptive text have been omitted.

IF Logic: chk(19, 1) == 1

Checks whether regime 1 is AI-controlled (1) or human-controlled (0).

SET Logic: exe(112, 10, 5)

Stores the numeric value 5 in Game Variable 10.

How to use the reference

A Check reads a numeric value; an Exec changes scenario data. Every command is identified by a numeric ID followed by its arguments in order. Omit unused trailing arguments.

Chk(19, TURN)

Returns 1 if the regime whose number is in TURN is controlled by the AI, otherwise 0.

Exe(112, 10, 5)

Sets Game Variable 10 to 5. Use numeric values and numeric variables when building IF Logic or SET Logic expressions.

Conventions: "#" means a numeric ID or slot; X and Y identify map hexes. Some commands reserve special numeric meanings for −1, 0, or 1, as documented in their parameter descriptions.

Only the numeric forms relevant to IF Logic and SET Logic are included. Obsolete, unsupported, internal and non-public functions are excluded.

Finding a command

The numeric Check and Exec directories below are sorted by function ID. Each entry includes its call form, numeric behavior and argument labels. Use Google Docs search (Ctrl+F) to jump directly to an ID or function name.

Check functions | Exec commands | SFType statistic selectors

CHECKS 001–050

All Checks in this directory can produce numeric results when used as described. For a StringList cell, the value stored in that cell must be numeric.

Chk(1) — CheckTurn

Returns: Returns the regime whose turn is currently active.

Arguments: None.

Chk(2, X, Y) — CheckHexOwner

Returns: Owner regime of hex (x,y); invalid map coordinates return -1.

Arguments: 1. X · 2. Y

Chk(3) — CheckRound

Returns: Returns the current round number.

Arguments: None.

Chk(4, REGIME) — CheckRegimeMorale

Returns: Base morale of the specified regime.

Arguments: 1. RegimeNr

Chk(5) — CheckWinner

Returns: Current global winner regime slot; the supplied regime parameter is ignored.

Arguments: None.

Chk(6, REGIMENR_QUESTION, RELATION_REGIMENR) — CheckWar

Returns: Whether the two regimes are at war (1=yes, 0=no).

Arguments: 1. RegimeNr in Question · 2. Relation with RegimeNr

Chk(7, REGIMENR_QUESTION, RELATION_REGIMENR) — CheckPeace

Returns: Whether the two regimes are at peace (1=yes, 0=no).

Arguments: 1. RegimeNr in Question · 2. Relation with RegimeNr

Chk(8, AREASLOT, AREACODE, REGIME, NOT_PEOPLENR) — CheckUnitsFrom

Returns: Total unit power points for a specified people type in an area-code region and regime.

Arguments: 1. AreaSlot · 2. AreaCode · 3. RegimeNr · 4. Not From PeopleNr

Chk(10) — CheckRandomPercent

Returns: Random integer from 1 through 100 (inclusive).

Arguments: None.

Chk(11, X, Y) — CheckLandscapeType

Returns: Landscape type ID at the specified hex.

Arguments: 1. X · 2. Y

Chk(12, X, Y) — CheckLandscapeSprite

Returns: Landscape sprite number at the specified hex.

Arguments: 1. X · 2. Y

Chk(13, X, Y, SLOT) — CheckSlot

Returns: Value of a specified area/hex slot at (x,y).

Arguments: 1. X · 2. Y · 3. Slot#

Chk(14) — CheckMapWidth

Returns: Maximum X/map-width coordinate for the current event map.

Arguments: None.

Chk(15) — CheckMapHeight

Returns: Maximum Y/map-height coordinate for the current event map.

Arguments: None.

Chk(16) — CheckYear

Returns: Year of the current game date; date-mode handling may return 0.

Arguments: None.

Chk(17) — CheckMonth

Returns: Month of the current game date; date-mode handling may return 0.

Arguments: None.

Chk(18) — CheckDay

Returns: Day of the current game date; date-mode handling may return 0.

Arguments: None.

Chk(19, REGIME) — CheckIfAI

Returns: Returns 1 if the specified regime is AI-controlled; returns 0 if it is human-controlled.

Arguments: 1. Regime#

Chk(20, REGIME) — CheckPolPts

Returns: Political/resource points currently held by the regime.

Arguments: 1. Regime#

Chk(21, REGIME) — CheckVP

Returns: Regime victory points using GetRegimeVP.

Arguments: 1. Regime#

Chk(22, YEAR, MONTH, DAY) — CheckDifferenceInDays

Returns: Day difference between an input year/month/day and the game's date.

Arguments: 1. Year · 2. Month · 3. Day

Chk(23, REGIME) — CheckSupplyNeeded

Returns: Total supply required for the selected regime.

Arguments: 1. Regime

Chk(24, X, Y) — CheckHQFor

Returns: Whether an HQ is present/eligible at the selected hex according to the event lookup.

Arguments: 1. X · 2. Y

Chk(27, REGIME_OWNER, CARD_NR) — CheckActionCard

Returns: Whether a regime currently holds the specified action card (1/0).

Arguments: 1. Regime Owner · 2. Card Nr

Chk(28, REGIME) — CheckDipBlock

Returns: Diplomatic block flag for a regime (1/0).

Arguments: 1. Regime#

Chk(29, REGIME) — CheckSleep

Returns: Sleep/inactive flag for a regime (1/0).

Arguments: 1. Regime#

Chk(30, REGIME_OWNER, CARD_NR) — CheckHistoricActionCard

Returns: Whether a card appears in the regime's historical action-card list (1/0).

Arguments: 1. Regime Owner · 2. Card Nr

Chk(31, AREASLOT, AREACODE, SFTYPE, REGIME) — CheckSFTypeInArea

Returns: Total quantity of a specified subformation type in an area region for a regime.

Arguments: 1. AreaSlot · 2. AreaCode (-1=all hexes) · 3. SFType · 4. Regime

Chk(32, AREASLOT, AREACODE, REGIME, LAND_ONLY) — CheckPowerPointsInArea

Returns: Total power points of units in an area-code region.

Arguments: 1. AreaSlot · 2. AreaCode · 3. Regime · 4. Land only (0=no check, 1=land theater only)

Chk(33, X, Y, SFTYPE, REGIME) — CheckSFTypeInXY

Returns: Total quantity of a specified subformation type at (x,y) for a regime.

Arguments: 1. X · 2. Y · 3. SFType (-1=all) · 4. Regime

Chk(34, REGIME, SFTYPE, ROUND, REPORT_0QTY_1POWER) — CheckRegimeKills

Returns: Recorded regime kills, filtered by type/round and reporting mode.

Arguments: 1. Regime · 2. SFType (-1=all, 1000-10XX = SFTypeGroups 0-XX) · 3. Round (-1=all rounds) · 4. Report 0=qty, 1=power pts

Chk(35, REGIME, SFTYPE, ROUND, REPORT_0QTY_1POWER) — CheckRegimeLosses

Returns: Recorded regime losses, filtered by type/round and reporting mode.

Arguments: 1. Regime · 2. SFType (-1=all, 1000-100X = SFTypeGroups 0-XX) · 3. Round (-1=all rounds) · 4. Report 0=qty, 1=power pts

Chk(36, REGIME, SFTYPE, ROUND, REPORT_0QTY_1POWER) — CheckRegimeProduction

Returns: Recorded regime production, filtered by type/round and reporting mode.

Arguments: 1. Regime · 2. SFType (-1=all) · 3. Round (-1=all rounds) · 4. Report 0=qty, 1=power pts

Chk(37) — CheckRegimeCount

Returns: Maximum regime index (RegimeCounter), not a human-style count.

Arguments: None.

Chk(38, X, Y, X2, Y2) — CheckIfNeighbour

Returns: Whether two hex coordinates are neighbours (1/0); invalid origin gives 0.

Arguments: 1. X · 2. Y · 3. X2 · 4. Y2

Chk(39, X, Y, X2, Y2) — CheckDistance

Returns: Distance between two hex coordinates; invalid origin returns 999.

Arguments: 1. X · 2. Y · 3. X2 · 4. Y2

Chk(40, X, Y, SFTYPE_GROUP, REGIME) — CheckSFTypeGroupInXY

Returns: Total quantity of a subformation group at a hex for a regime.

Arguments: 1. X · 2. Y · 3. SFType Group (-1=all) · 4. Regime

Chk(42, REGIME) — CheckAIThinkSpeed

Returns: Returns the specified regime's AI production bonus (ProdBonus).

Arguments: 1. Regime

Chk(43) — CheckTotalUnits

Returns: Maximum unit index (UnitCounter), not necessarily number of live units.

Arguments: None.

Chk(44, UNIT) — CheckUnitOwner

Returns: Owner regime of a unit slot; -1 input returns -1.

Arguments: 1. Unit Nr

Chk(45, UNIT) — CheckUnitHQ

Returns: Parent HQ unit slot of a unit.

Arguments: 1. Unit Nr

Chk(46, UNIT) — CheckUnitX

Returns: Current X coordinate of a unit.

Arguments: 1. Unit Nr

Chk(47, UNIT) — CheckUnitY

Returns: Current Y coordinate of a unit.

Arguments: 1. Unit Nr

Chk(48, UNIT) — CheckUnitHistoricalID

Returns: Historical unit ID attached to a unit; -1 if none.

Arguments: 1. Unit Nr

Chk(49, UNIT) — CheckUnitIsHQ

Returns: Whether a unit is an HQ (1/0).

Arguments: 1. Unit Nr

Chk(50, REG1, REG2) — CheckRegimeRelation

Returns: Stored diplomatic relation code between two regimes.

Arguments: 1. Reg1 · 2. Reg2

Note: Returns 0 for war, 1 for peace, 2 for alliance.

CHECKS 051–099

All Checks in this directory can produce numeric results when used as described. For a StringList cell, the value stored in that cell must be numeric.

Chk(51, UNIT, HQ_NR) — CheckUnitInHQChain

Returns: Whether one unit belongs to another unit's HQ command chain (1/0).

Arguments: 1. Unit Nr · 2. HQ Nr

Chk(52, UNIT) — CheckUnitCommanderStaffPercent

Returns: Commander staff as a percentage of HQ command power (0 for non-HQ).

Arguments: 1. Unit Nr

Chk(53, UNIT) — CheckUnitHQPower

Returns: HQ power as calculated by Gethqpow.

Arguments: 1. Unit Nr

Chk(54, HISTORICAL_ID) — CheckFirstUnitWithHistoricalID

Returns: First active unit slot matching a historical unit index; -1 if absent.

Arguments: 1. Historical ID

Chk(55, X, Y) — CheckLocTypeXY

Returns: Primary location type at (x,y); -1 if no primary location.

Arguments: 1. X · 2. Y

Chk(56, X, Y) — CheckStrucPtsXY

Returns: Primary location structural points at (x,y); 0 if absent.

Arguments: 1. X · 2. Y

Chk(57, X, Y, REGIME) — CheckClosestCommanderUnit

Returns: Unit slot of closest commander/HQ unit for map coordinate and regime.

Arguments: 1. X · 2. Y · 3. Regime

Chk(58, STRINGLIST_ID, ROW, COL) — CheckStringList

Returns: Reads the numeric contents of a StringList cell. Use only if the selected cell contains a number; text-valued cells are outside the scope of IF/SET Logic.

Arguments: 1. Stringlist ID · 2. Row · 3. Col

Numeric use only: if the cell contains letters or other text, do not use the returned value in IF/SET Logic calculations.

Chk(60, REGIME_NR, VAR_NR) — CheckRegimeVar

Returns: Read a numbered regime variable/RegimeSlot.

Arguments: 1. Regime Nr · 2. Var Nr

Chk(61, UNIT_NR, TEMPVAR_NR, USE_HISTORICAL_UNIT) — CheckHistoricalUnitTempVar

Returns: Retrieve a commander/historical-unit temporary variable or AI list, using selector mode.

Arguments: 1. Unit Nr (or -1 to use hist.) · 2. TempVar Nr (1-5), 6=AiZoneListNr · 3. Use Historical Unit ID

Chk(63, UNIT_NR, OVERRULE_HIS_UNIT, MODE, OPT_CARD_EV) — CheckCommanderXP

Returns: Commander XP, deck-card or auto-event information, selected by mode.

Arguments: 1. Unit Nr (or -1) · 2. Overrule His Unit ID (or -1) · 3. 0=Xp, 1=Total Cards, 2=Specific Card, 3=Total Ev, 4=Spec Ev · 4. Opt: card or ev#

Chk(64, STRINGLIST_ID) — CheckStringListRows

Returns: Maximum row index of a StringList; -1 if list absent.

Arguments: 1. StringList ID

Chk(65, STRINGLIST_ID) — CheckStringListCols

Returns: Maximum column index of a StringList; -1 if list absent.

Arguments: 1. StringList ID

Chk(66) — CheckStringListLastID

Returns: Canonical/current StringList ID returned through ID resolution.

Arguments: None.

Chk(67, UNIT) — CheckUnitRdn

Returns: Average unit readiness.

Arguments: 1. Unit#

Chk(68, UNIT, SFTYPE_GROUP, SPECIFIC_SFTYPE) — CheckUnitSFType

Returns: Number/quantity of a specific subformation type within a unit.

Arguments: 1. Unit # · 2. SFType Group (-1 for none) · 3. Specific SFType (-1 for none)

Note: Returns the number of individuals in this unit.

Chk(71, REG) — CheckRegimeUnitsMorale

Returns: Power-weighted average morale of a regime's units.

Arguments: 1. Reg

Note: Returns the average morale of all units on map for this regime.

Chk(72, REG) — CheckRegimeUnitsExperience

Returns: Power-weighted average experience of a regime's units.

Arguments: 1. Reg

Chk(73, UNIT) — CheckUnitTempOwner

Returns: Unit's effective/temporary owning regime, with -1 for inactive predefined units.

Arguments: 1. Unit Nr

Note: Returns the real owner. If this is a subregime it will actually give back the subregime regime # instead of the uberregime which the normal CheckUnitOwner will return.

Chk(75, UNIT) — CheckUnitPower

Returns: Absolute power points of the selected unit.

Arguments: 1. Unit Nr

Note: Returns the real total power points in the unit.

Chk(76, X, Y) — CheckSpecialType

Returns: Hex special-type value at (x,y).

Arguments: 1. X · 2. Y

Chk(77, X, Y) — CheckSpecialSprite

Returns: Hex special sprite number at (x,y).

Arguments: 1. X · 2. Y

Chk(78, UNIT) — CheckUnitSelectable

Returns: Unit selectable flag (1/0).

Arguments: 1. Unit Nr

Note: Returns 0 if not selectable. returns 1 if selectable.

Chk(79, MINIMUM, X, Y, MAX_DISTANCE) — DistanceClosestEnemyHex

Returns: Distance to closest relevant enemy hex, subject to AI value and distance limits.

Arguments: 1. Minimum (AI)VP value? 0=any · 2. from X · 3. from Y · 4. Max distance (-1=no maximum)

Note: Returns the distance. Max distance can be set to keep processing time down if necc.

Chk(80, MAX_DISTANCE, X, Y, DISTANCE_DIVIDER_MODIFI) — CheckEnemyTroopsCloseBy

Returns: Enemy troop/power presence around a hex within supplied distance constraints.

Arguments: 1. Max Distance · 2. from X · 3. from Y · 4. Distance divider modifier (0=none)

Note: Max distance sets circle in which too look. Distance divider modifier set by what the power points of a unit are divided. if 0 they are not divided. if 1 they are divided by distance. if 2 by double distance.. etc.

Chk(81, UNIT) — CheckUnitMor

Returns: Average morale of the selected unit.

Arguments: 1. Unit#

Note: Keep in mind this is an average of the subformations.

Chk(83, SFTYPE, SFTYPE_TARGETGROUP, MODE) — CheckSFTypePower

Returns: Attack power or defensive attack power against a target group, selected by mode.

Arguments: 1. SFType# · 2. SFType Targetgroup · 3. 1=attack score, 2=defend score

Chk(85, SFTYPE, SFTYPE_TARGETGROUP, MODE) — CheckSFTypeHP

Returns: Attacking or defending hit points versus target group, selected by mode.

Arguments: 1. SFType# · 2. SFType Targetgroup · 3. 1=attack hp, 2=defend hp

Chk(86, SFTYPE) — CheckSFTypeMoveType

Returns: Movement type code of a subformation type.

Arguments: 1. SFType#

Chk(87, SFTYPE) — CheckSFTypeMoveRedux

Returns: Movement reduction value of a subformation type.

Arguments: 1. SFType#

Chk(88, SFTYPE, RESEARCH) — CheckSFTypeModelLastState

Returns: Last model state for a particular model index.

Arguments: 1. SFType# · 2. Research#

Chk(92, SFTYPE, MODE) — CheckSFTypeFuelUse

Returns: Fuel use for movement or attack by subformation type, depending on selector.

Arguments: 1. SFtype# · 2. 1=ForMovement, 2=ForCombat

Chk(93, NR) — CheckGameVar

Returns: Value stored in a numbered global GameSlot.

Arguments: 1. Nr

Chk(94, NR) — CheckTempVar

Returns: Value stored in a numbered TempVar.

Arguments: 1. Nr

Chk(95, REGIME) — CheckRegimePeople

Returns: People ID assigned to the specified regime.

Arguments: 1. Regime

Chk(96, SFTYPE, VAR) — CheckSFTypeVar

Returns: Value of a subformation-type variable.

Arguments: 1. SFType · 2. Var#

Chk(97, SFTYPE) — CheckSFTypeItem

Returns: Model item-type ID associated with a subformation type.

Arguments: 1. SFType#

Chk(98, X, Y) — CheckHexVP

Returns: Hex victory-point value (uses currently selected map, not necessarily EventMap).

Arguments: 1. X · 2. Y

Chk(99, REGIME, TOO_REGIME) — CheckDipOffer

Returns: Regime's diplomatic offer toward another regime.

Arguments: 1. From Regime · 2. Too Regime

CHECKS 101–150

All Checks in this directory can produce numeric results when used as described. For a StringList cell, the value stored in that cell must be numeric.

Chk(101, SLOT, VALUE) — CheckDefinedAreaID

Returns: Find a defined-area ID from area slot and area code; -1 if absent.

Arguments: 1. Slot#(0-9) · 2. Value#

Note: Returns first Defined area with that slot&value coupled to it.

Chk(102, X, Y) — CheckHexPeople

Returns: People ID of the primary location on a hex; -1 when absent.

Arguments: 1. X · 2. Y

Note: Returns people number of location here. if any. if not returns -1.

Chk(104) — CheckNonSleepingRegimesToCome

Returns: Find a non-sleeping regime relevant to hex coordinates.

Arguments: None.

Note: Returns the number of regimes that are still to be played in this round that are not put to sleep.

Chk(108, VAR_X, VAR_Y) — CheckCalcGetPercent

Returns: Calculate a percent-related value using the check's integer inputs.

Arguments: 1. Var X · 2. of Var Y

Note: Returns the percentage that var X makes up of var Y.

Chk(109) — CheckLocCounter

Returns: Maximum location index (LocCounter).

Arguments: None.

Note: Returns the number of locations. remember counting starts at 0.

Chk(110, LOCATION) — CheckLocX

Returns: X coordinate of a location slot; -1 for a -1 location.

Arguments: 1. Location#

Chk(111, LOCATION) — CheckLocY

Returns: Y coordinate of a location slot; -1 for a -1 location.

Arguments: 1. Location#

Chk(113, REGIME, FUEL_TYPE) — CheckMaxFuelUse

Returns: Total fuel-use demand by regime and fuel type.

Arguments: 1. Regime (-1=all) · 2. Fuel Type (-1=all)

Chk(117, X, Y) — CheckHexLocNr

Returns: Primary location slot at hex (x,y).

Arguments: 1. X · 2. Y

Note: Returns -1 if no Location in hex. 0 or higher if present.

Chk(118, X, Y) — CheckHexUnitCounter

Returns: Hex UnitCounter (maximum zero-based unit-list index).

Arguments: 1. X · 2. Y

Note: Returns -1 if no units. Returns 0 is one unit (in slot 0). Returns 1 if two units present (one in slot 0, one in slot1), etc.

Chk(119, X, Y, UNIT_SLOT) — CheckHexUnit

Returns: Unit ID in a particular index of a hex's unit list.

Arguments: 1. X · 2. Y · 3. Unit Slot

Note: You should use CheckHexUnitCounter to determine the number of units on the hex. Start with slot=0!.

Chk(121, UNIT, CHECKMODE0_1_2) — CheckUnitStaffPercent

Returns: Unit staff percentage or corresponding combat/morale staff modifiers by mode.

Arguments: 1. Unit Nr · 2. CheckMode=0,1,2

Note: Return the staff % of the unit... you might want to use this check in combination with CheckUnitHqPower since distance is not checked for in this event. Check Mode 0 returns staff%. Check Mode 1 returns combat mod%, Check Mode 2 returns morale mod%.

Chk(122, SFTYPE) — CheckSFTypeRatio

Returns: Ratio/multiplier for a subformation type.

Arguments: 1. SFType#

Chk(124, X, Y, DIRECTION) — CheckRoadType

Returns: Road-type information at a hex, using its edge/selection parameter.

Arguments: 1. X · 2. Y · 3. Direction

Note: Direction: 0=n, 1=ne, 2=se, 3=s, 4=sw, 5=nw. Use direction -1 to test for all directions: first type found is returned. -1 is returned if none.

Chk(125, X, Y, DIRECTION) — CheckRiverType

Returns: River-type information at a hex, using its edge/selection parameter.

Arguments: 1. X · 2. Y · 3. Direction

Note: Direction: 0=n, 1=ne, 2=se, 3=s, 4=sw, 5=nw. Use direction -1 to test for all directions: first type found is returned. -1 is returned if none.

Chk(126, X, Y, DIRECTION) — CheckBridge

Returns: Whether a bridge is present on the specified hex edge (1/0).

Arguments: 1. X · 2. Y · 3. Direction

Note: Direction: 0=n, 1=ne, 2=se, 3=s, 4=sw, 5=nw. Use direction -1 to test for all directions. Returns 0 for no bridge and 1 for yes bridge.

Chk(127, HIS_UNIT_ID, TYPE) — CheckHisVar

Returns: Value of a numbered historical-unit variable (0 if not found).

Arguments: 1. His Unit ID# · 2. Type#

Note: Returns value if type is present. If type is not present it returns 0.

Chk(128, UNIT, INCLUDE_LAND_THEATER, INCLUDE_SEA_THEATER, INCLUDE_AIR_THEATER) — CheckUnitRecon

Returns: Reconnaissance value of a specified unit.

Arguments: 1. Unit# · 2. Include land theater 0=no, 1=yes · 3. Include sea theater 0=no, 1=yes · 4. Include air theater 0=no, 1=yes

Note: Returns total recon points in the unit.

Chk(129, X, Y, SPECIFIC_ROAD_TYPE) — CheckRoadTypeSpecific

Returns: Whether a specific road type is found on a hex (1/0).

Arguments: 1. X · 2. Y · 3. Specific Road Type

Note: Check is specied roadtype is present in this hex. Returns 1 if is. Returns 0 if not.

Chk(130, UNIT) — CheckHistoricalUnitAverageXP

Returns: Average experience of a historical unit, weighted by its subformations.

Arguments: 1. Unit#

Note: Will look up other units on map with the same historical unit and return average xp.

Chk(131, STRINGLIST_ID, COL_USED_WEIGHT) — CheckRandomRowStringList

Returns: Random row number of a StringList, using a weight column; -1 if unavailable.

Arguments: 1. Stringlist ID · 2. Col used as weight (-1=none)

Chk(132, STRINGLIST_ID, COL, FIND_VALUE) — CheckFindInStringList

Returns: First StringList row where the selected column matches a supplied numeric value; -1 if no match.

Arguments: 1. Stringlist ID · 2. In Col · 3. Find Value

Note: Returns row# where value is found or -1 if not.

The searched value can be stored as text internally, but it must be numeric for this guide.

Chk(133) — CheckTotalHistoricalUnits

Returns: Maximum historical-unit index (HistoricalUnitCounter).

Arguments: None.

Chk(134, SLOT) — CheckGetHistoricalUnitID

Returns: Stable historical-unit ID for the provided slot; -1 for negative slot.

Arguments: 1. Slot#

Note: Keep in mind that this function should be used to scroll through all historical units using a looper from 0 to CheckTotalHistoricalUnits. Use this function to get the actual ID to be used in other functions.

Chk(135, HISTORICAL_UNIT_ID) — CheckGetHistoricalUnitRegime

Returns: Temporary owner/regime of a historical unit, or -1.

Arguments: 1. Historical Unit ID#

Chk(136, HQ_LEVEL_CHECK, TROOP_PEOPLE, HQ_PEOPLE, REGIME) — CheckPeopleUnderHQ

Returns: Percentage of specified people under HQ command using HQ level and regime filters.

Arguments: 1. HQ level to check · 2. Troop people · 3. HQ people · 4. regime#

Note: A complicated function. HQ level 1 is HQs with historical unit set to Corps. HQ level 2 is only HQs with historical HQ set to Army, 3=armygroup, 4=high command.. Function returns % of total power points of people on map is under a hq whose historical unit people is set to hqppl. This function could for example be used to check if axis minor are under wehrmacht HQs or under their own.

Chk(137, HISTORICAL_UNIT_ID) — CheckGetHistoricalUnitPeople

Returns: People ID associated with a historical unit, or -1.

Arguments: 1. Historical Unit ID#

Chk(138, HISTORICAL_UNIT_ID) — CheckGetHistoricalUnitType

Returns: Historical unit type code, or -1.

Arguments: 1. Historical Unit ID#

Note: Returns the type. 5=corps, 6=army, etc.

Chk(139, HISTORICAL_UNIT_ID) — CheckGetHistoricalUnitOfficer

Returns: Whether a historical unit has a commander name (1/0).

Arguments: 1. Historical Unit ID#

Note: returns 1 if officer, returns 0 if not. Basicly the length of officer name is checked.

Chk(141, VALUE1, VALUE2) — CheckMin

Returns: Minimum of two supplied integers.

Arguments: 1. Value1 · 2. Value2

Note: Returns the lowest of the 2 values entered.

Chk(142, VALUE1, VALUE2) — CheckMax

Returns: Maximum of two supplied integers.

Arguments: 1. Value1 · 2. Value2

Note: Returns the highest of the 2 values entered.

Chk(144, HISTORICAL_UNIT_ID) — CheckGetHistoricalSubparts

Returns: Number of subparts associated with a historical unit.

Arguments: 1. Historical Unit ID#

Chk(145, REGIME) — CheckTotalSupplyUse

Returns: Total supply usage of a specified regime.

Arguments: 1. Regime #

Note: Returns the BasicSupplyUse of all troops on the map added up.

Chk(146, UNIT) — CheckUnitAP

Returns: Available action points calculated for a unit.

Arguments: 1. Unit#

Note: Keep in mind this is the lowest AP subformations.

Chk(147, REGIME) — CheckPbemPlayer

Returns: PBEM player field of a regime.

Arguments: 1. Regime#

Note: Returns either 1 for pbem player 1 (challenger), 2 for pbem player 2(opponent) or 0 if not set (not a started pbem++ game for example).

Chk(148) — CheckDrawGame

Returns: Whether game was drawn, using the game's draw status (1/0).

Arguments: None.

Note: Returns 0 if not. Returns 1 if this is a draw game.

Chk(149, REGIME, CATEGORY, COLOR) — CheckTotalActioncards

Returns: Number of action cards, optionally filtered by regime/category.

Arguments: 1. Regime # · 2. Category # (-1=all) · 3. Color # (-1=all)

Note: Returns the total number of action cards the regime has in hand. However if you set regime=-1 it returns the top action card slot in the actioncard list. This is usefull if your manually adding actioncards through events.

Chk(150, REGIME) — CheckAICombatMod

Returns: Returns the configured AI combat bonus/difficulty value for the specified regime.

Arguments: 1. Regime

Note: Gives back the AI combat modifier. 0=none, >0 is the percentual bonus. (This is the AI difficulty in the scenario setup screen).

CHECKS 151–200

All Checks in this directory can produce numeric results when used as described. For a StringList cell, the value stored in that cell must be numeric.

Chk(151, UNR) — CheckIsUnitAPreDef

Returns: PreDef index of a unit, or -1 if none.

Arguments: 1. Unr#

Note: Returns -1 if not a predef. Returns the predef ID # if it is a predef.

Chk(152, RULEVAR, MULITPLY_BY_100) — CheckRuleVar

Returns: Numbered RuleVar, either integer-truncated or multiplied by 100 and truncated.

Arguments: 1. RuleVar# · 2. Mulitply by 100? 0=no 1=yes

Note: Multiply option is there because some rulevar have single values like 0.4 or 0.03. And the scripting sytem can only handle integers.

Chk(153, UNIT, ONLY_IF_STAFF) — CheckUnitXP

Returns: Unit XP, optionally conditioned on staff.

Arguments: 1. Unit# · 2. Only if staff (0=no,1=yes)

Note: Keep in mind that it returns the highest XP found in any subformation.

Chk(156, UNIT) — CheckUnitHQChanged

Returns: Whether unit HQ assignment has changed (1/0).

Arguments: 1. Unit Nr

Note: Returns 1 if the unit HQ has been changed in this turn. Returns 0 if not.

Chk(157, HIS_UNIT_ID, CHECK_AIR_THEATER, CHECK_NAVY_THEATER) — CheckHistoricalUnitAirAndNavy

Returns: Whether historical unit includes selected air or naval subformations (1/0).

Arguments: 1. His Unit ID# · 2. Check for air theater troops? 0=no,1=yes · 3. Check for navy theater troops? 0=no,1=yes

Note: Returns 1 if those troops are present. 0 if not.

Chk(158, X, Y, PROPERTY) — CheckForLocationProperty

Returns: Returns 1 if the hex has the requested facility (1=airfield, 2=port with adjacent sea); otherwise 0.

Arguments: 1. X · 2. Y · 3. Property (1=airfield, 2=port)

Chk(182, SFTYPE) — CheckVisible

Returns: Whether a subformation type is marked visible (1/0).

Arguments: 1. SFType#

Note: Returns 1 if visible (dont show in list=false), returns 0 if not (dont show in list=true). Invisible SFTypes are used as base models in the Intermediate TroopType editor.

Chk(183) — CheckSFTypeCounter

Returns: Maximum subformation-type index (SFTypeCounter).

Arguments: None.

Chk(184, SFTYPE, MODE) — CheckSFTypeTheaterOrGroup

Returns: Theater or unit-group code for a subformation type, according to selector.

Arguments: 1. SFType# · 2. 1=Check Theater, 2=CheckUnitGroup

Note: Theater codes are: 0=land, 1=sea, 2=air.

Chk(185, HISTORICAL_UNIT_ID) — CheckHistoricalUnitMaster

Returns: Model-master historical ID of a historical unit; -1 if not modeled.

Arguments: 1. Historical Unit ID

Note: Returns the historical unit model ID.

Chk(187, UNIT, ACTIVITY_DATA_TYPE) — CheckUnitActivity

Returns: Offensive combat, defensive combat or movement activity for a unit; see mode list.

Arguments: 1. Unit Nr · 2. Activity Data Type

Note: These stats are reset after lateTurn events are executed. Activity Data Types are 1=Offensive combat AP spent, 2=Defensive combat AP spent, 3=Move AP spent, 4=Off+Def AP spent, 5=Off+Def+Move AP spent. 6=Did unit spent move AP or moved (0=no, 1=yes).

Chk(189, TEMPSTRING) — CheckTempString

Returns: Read a RealTempString slot.

Arguments: 1. TempString #

Chk(190) — CheckPopupSkipped

Returns: Whether action-card preselection popup was skipped (1/0).

Arguments: None.

Note: Returns 1 if popup skipped otherwise returns 0.

Chk(191, UNIT_NUMBER, PLAYER_NUMBER) — CheckForUnitVisibleToPlayer

Returns: Whether a unit can be seen by a player; may return a recon/visibility value, not strictly Boolean.

Arguments: 1. Unit Number · 2. Player Number

Note: Returns 1,2,3 if unit is visible. Returns 0 if it is not. 1=?mark counter, 2=fuzzy info, 3=exact strength known.

Chk(192) — CheckFOW

Returns: Whether fog of war is enabled (1/0).

Arguments: None.

Chk(193, X, Y) — CheckMoveOriginX

Returns: X coordinate of cached movement origin at coordinate (x,y); -1 if off-map.

Arguments: 1. X · 2. Y

Note: You should only call this function if ExecSetMoveMatrix or ExecSetSupplyMatrix has been called before it. Any hex that has a moveable score (<9999) and that is not the scenario hex of the matrix called will have an origin. These 2 functions (x and y) will return that origin coordinate. -1 is returned if no origin hex is available.

Chk(194, X, Y) — CheckMoveOriginY

Returns: Y coordinate of cached movement origin at coordinate (x,y); -1 if off-map.

Arguments: 1. X · 2. Y

Note: You should only call this function if ExecSetMoveMatrix or ExecSetSupplyMatrix has been called before it. Any hex that has a moveable score (<9999) and that is not the scenario hex of the matrix called will have an origin. These 2 functions (x and y) will return that origin coordinate. -1 is returned if no origin hex is available.

Chk(195, CARD_SLOT) — CheckActionCardPPCost

Returns: Political-point cost of a card; -1 for invalid card slot.

Arguments: 1. Card slot #

Note: Returns the PP cost of the action card specified.

Chk(196, ID) — CheckGetHistoricalUnitSlot

Returns: Find historical-unit slot from its stable ID; -1 if missing.

Arguments: 1. ID#

Note: Supply the historical unit ID# and get the slot# in return. If not found it will return -1.

Chk(197, CARD_SLOT, TEMPVAR_0_1) — CheckActionCardTempVar

Returns: Action card's TempVar0 or TempVar1, selected by 0 versus nonzero.

Arguments: 1. Card slot # · 2. Tempvar 0 or 1

Note: Returns the tempvar number selected.

Chk(199, ID) — CheckGetCommanderPoolValue

Returns: Whether a commander/historical unit belongs to the commander pool (1/0).

Arguments: 1. ID#

Note: Returns 1 if in pool. Returns 0 if not.

Chk(200, NUMBER, DIVIDER) — CheckModulo

Returns: Computes (value + divisor) Mod divisor; returns 0 when divisor is nonpositive.

Arguments: 1. Number · 2. Divider

Note: Return (number) modulo (divider)... For example 4 modulo 3 = 1.

CHECKS 201–250

All Checks in this directory can produce numeric results when used as described. For a StringList cell, the value stored in that cell must be numeric.

Chk(201, AREASLOT, STARTVAL, ENDVAL, OWNER) — CheckSlotOwnerPercentage

Returns: Percentage of hexes in an area-slot range belonging to a given regime.

Arguments: 1. AreaSlot# · 2. Startval (>=) · 3. Endval (<=) · 4. Owner

Note: Returns a percentage of the specified areaslots between start and endvalue occupied by owner regime slot#.

Chk(202, AREASLOT, STARTVAL, ENDVAL, MAX_DIST) — CheckSlotNotInAIList

Returns: Pick a reachable unassigned area-code target for AI HQ assignment; -1 if none.

Arguments: 1. AreaSlot# · 2. Startval (>=) · 3. Endval (<=) · 4. Max Dist

Note: Within areaslot returns a value out of the range between start and endvalue occupied by current regime owner slot# closest to SetAreaXY specified. Returns -1 if nothing found. Function only looks within supply range of SetAreaXY and only within MaxDist.

Chk(203, AREASLOT, VALUE, ARG3) — CheckSlotValueFirstXY

Returns: First (x,y) or regime for an area-code value, chosen by mode; -1 if absent.

Arguments: 1. AreaSlot# · 2. Value · 3. What? 1=x,2=y,3=reg

Note: Within areaslot returns the x,y coordinate first found with the value given (x or y). Only values >= 1 work.

Chk(204, NUMBER) — CheckAbs

Returns: Absolute value of an integer.

Arguments: 1. Number

Note: Return the absolute of the number passed.

Chk(206, STRINGLIST_ID, COL_USED_WEIGHT, COL_USED_BLOCK) — CheckRandomRowStringList2

Returns: Weighted random StringList row, respecting a blocked-column constraint.

Arguments: 1. Stringlist ID · 2. Col used as weight (-1=none) · 3. Col used as block (-1=none)

Note: Concerning the block column. <= 0 will be considered NOT blocked. >= 1 will be considered blocked. Returns -1 if no suitable row found.

Chk(209, UNIT) — CheckUnitEntrench

Returns: Average entrenchment of a selected unit.

Arguments: 1. Unit#

Note: Keep in mind this is an average of the subformations.

Chk(211, UNIT) — CheckUnitIntegrity

Returns: Calculated unit-integrity score.

Arguments: 1. Unit#

Note: Returns a percentage between 0-100.

Chk(212, UNIT, MODE) — CheckUnitSupplyConsumption

Returns: Current supply consumption or storage/fuel capacity for a unit by mode.

Arguments: 1. Unit# · 2. Mode

Note: Mode<=1 : Returns a percentage between 0-100. Mode 2: Returns missing supply, Mode 3: Returns missing fuel.

Chk(218, NUMBER) — CheckSqrt

Returns: Square root of input, converted to integer.

Arguments: 1. Number

Note: Returns the square root of the number.

Chk(219, NUMBER1, NUMBER2) — CheckRandomNumber

Returns: Random integer between inclusive lower and upper limits.

Arguments: 1. Number1 · 2. Number2

Note: Returns a random number including and between number1 and number2.

Chk(222, NUMBER_1, NUMBER_2) — CheckPercentage

Returns: Percentage as integer: 100 × part ÷ total; 0 for nonpositive total.

Arguments: 1. Number 1 (part) · 2. Number 2 (total)

Note: Returns how much percentage points number 1 if of number 2.

Chk(223, X, Y, DIRECTION, MODE) — CheckNeighbour

Returns: Get neighbouring hex X or Y coordinate by direction/mode; -1 if off-map.

Arguments: 1. X · 2. Y · 3. Direction · 4. Mode

Note: Direction=1-6. Mode=1 returns X. Mode=2 returns Y. Returns -1 if neighbour is off-map.

Chk(226, NUMBER, NUMBER_SQUARE_ROOTS, MULTIPLY_RESULT_BY) — CheckSqrt2

Returns: Apply square root repeatedly, multiply by scale, and convert to integer.

Arguments: 1. Number · 2. Number square roots · 3. Multiply result by

Note: Returns the square root of the number. Allow for betting calculation of floating point numbers.

Chk(234, VALUE_STRING, MULTIPLIER, NUMBER_COMMA_SHIFTS) — CheckMultiplyByVerySmallNumber

Returns: Multiply a long integer by a decimal factor encoded as integer and power-of-ten divisor.

Arguments: 1. Value as string · 2. Multiplier · 3. Number of comma shifts

Note: Value as string allows you to work with long numbers. Number of comma shifts is applied to multiplier and then the value is multiplied by the multiplier. For exame value=1000, multiplier=5 and comma shift 2 results in 0.05*1000=50. Maximum 10 comma shifts.

Chk(235, NUMBER_A, DIVIDE_BY_NUMBER, NUMBER_SQUARE_ROOTS, MULTIPLY_BY_NUMBER) — CheckSqrt3

Returns: Square root of a quotient repeatedly, multiply by scale, and convert to integer.

Arguments: 1. Number a · 2. Divide by number b · 3. Number of square roots · 4. Multiply by number c

Note: Returns (the square root of (the number a / number b)) multiplied by number c. Allow for betting calculation of floating point numbers.

Chk(236, NUMBER) — CheckPowerLong

Returns: The square of the supplied integer, written as decimal digits. Values beyond normal integer range may not work in IF/SET Logic arithmetic.

Arguments: 1. Number

Note: Returns the power of the number as as string, allowing you to work with long numbers.

Chk(237, STRINGLIST_ID, COL, MUST_HAVE_VALUE, ADD_UP_VALUES) — CheckStringListTotal

Returns: Sum numeric values in a StringList column, optionally filtered by another column.

Arguments: 1. Stringlist ID · 2. Col# · 3. Must have value · 4. Add up values in col2#

Note: If you put -1 for Col# than all values will be added up.

Chk(238, REGIMESLOT) — CheckRegimeID

Returns: Stable ID of a regime slot.

Arguments: 1. RegimeSlot#

Chk(240, LOCATION_ID) — CheckLocationSlot

Returns: Find a location slot by stable location ID; -1 if missing.

Arguments: 1. Location ID

Chk(241, REGIME_ID) — CheckRegimeSlot

Returns: Find a regime slot by stable regime ID; -1 if missing.

Arguments: 1. Regime ID

Chk(242, STRINGLIST_ID, COL, FIND_VALUE, COL_MUST_HAVE) — CheckFindInStringList2

Returns: First StringList row matching the supplied numeric column value and additional numeric filter; -1 if none.

Arguments: 1. Stringlist ID · 2. In Col · 3. Find Value · 4. Col # must have value >0

Note: Returns row# where value is found or -1 if not.

Search for numeric values only; text-based searches are excluded from this reference.

CHECKS 251–264

All Checks in this directory can produce numeric results when used as described. For a StringList cell, the value stored in that cell must be numeric.

Chk(251, SFTYPE, TARGETGROUP, TYPE_1FAVSCORE_REGULAR) — ChecSFTypeFav

Returns: Favorite regular/artillery target rating for SF type and target group, selected by mode.

Arguments: 1. SfType # · 2. Targetgroup# · 3. Type 1=favscore for regular attack, 2=for artillery attack

Chk(252, SFTYPE, TARGETGROUP) — ChecSFTypeArtPower

Returns: Artillery attack power against a target group for a subformation type.

Arguments: 1. SfType # · 2. Targetgroup#

Chk(253, SFTYPE, STATNUMBER) — ChecSFTypeStat

Returns: Read one of 42 subformation-type statistic fields by property selector.

Arguments: 1. SfType # · 2. StatNumber

Note: Stat Numbers: 1=weight, 2=carrycap, 3=canDoParadrop(0/1), 4=EP, 5=BlowBridgPts, 6=BasicSupNeed, 7=SupCarry, 8=ReconPts, 9=HidePts, 10=AntiSupplyPts, 11=AntiSupplyRange, 12=AnitSupplySea, 13=InitiativeAtt, 14=InitiativeDef, 15=Attacks, 16=StackPts, 17=RearArea(0/1), 18=ArtRange, 19=Fav.Target tries, 20=AA-range, 21=HitKill%, 22=HitRetreat%, 23=ArtilleryModSfTypeNr, 24=maxAttacked, 25=moveRedux.

Chk(254, SFTYPE, LANDSCAPETYPE, MODE) — ChecSFTypeLandscapeMod

Returns: Returns the subformation type's landscape combat modifier as a percentage (1=attack, 2=defense).

Arguments: 1. SfType # · 2. LandscapeType# · 3. Mode (1=attack, 2=defense)

Chk(257, X, Y) — CheckHexLocNr2

Returns: Secondary location slot at a hex (Location2).

Arguments: 1. X · 2. Y

Note: Only use of Supply Bases and Sources. Returns -1 if no Location in hex. 0 or higher if present.

Chk(258, X, Y) — CheckLocTypeXY2

Returns: Secondary location type at a hex (Location2); -1 if missing.

Arguments: 1. X · 2. Y

EXECS 001–049

Execs modify scenario state or produce a defined event action.

Exe(1, REGIME, PP) — ExecGiveRegimePP

Effect: Add political/resource points to a regime.

Arguments: 1. RegimeNr · 2. PP

Exe(5, REGIME, MUTATION_PTS) — ExecAddRegimeMorale

Effect: Add an amount to a regime's base morale.

Arguments: 1. RegimeNr · 2. Mutation +/- pts

Exe(6, UNITPREDEF, X, Y, REGIME) — ExecAddUnit

Effect: Create/deploy a predefined unit at (x,y) for a regime; may interact with battle/ownership logic.

Arguments: 1. UnitPreDef# · 2. X · 3. Y · 4. Regime#

Exe(8, REGIME, SLEEP1_AWAKE0) — ExecSetSleep

Effect: Set a regime's sleeping/inactive state.

Arguments: 1. Regime# · 2. Sleep=1, Awake=0

Exe(9, LOSING_REGIME, ADDED_REGIME, INCLUDING_UNITS_1YES) — ExecJoinRegime

Effect: Merge one regime into another, optionally including units.

Arguments: 1. Losing Regime# · 2. Added to Regime# · 3. Including Units 1=yes,0=no

Exe(12) — BlockEvent

Effect: Block or unblock the currently executing event; first parameter comes from event context.

Arguments: None.

Exe(14, AREASLOT, AREACODE, REGIME, UNITS_FLEE_TOO) — ExecJoinArea

Effect: Transfer ownership of an area-code region, with optional unit transfer rules.

Arguments: 1. Areaslot# (0-9) · 2. Areacode · 3. To Regime# · 4. Units flee too areacode (-1=destr,-2=keephex)

Note: All units not allied with the new owner will be either flee or be destroyed or removed or keep hex.

Exe(15, REGIME, PEOPLE) — ExecDisbandUnits

Effect: Disband units of a regime, optionally filtered by people.

Arguments: 1. Regime# · 2. People#

Exe(16, REGIME) — ExecChangeDipBlock

Effect: Switch a regime's diplomatic block state.

Arguments: 1. Regime#

Exe(17, X, Y, DAMAGE_PTS) — ExecDoStructuralDamage

Effect: Apply structural damage at hex coordinates.

Arguments: 1. x · 2. y · 3. damage pts

Note: If you give negative ammount of points it is handled as a repair.

Exe(18, X, Y, NEW_LT) — ExecSetLandscape

Effect: Change the landscape type of a hex.

Arguments: 1. X · 2. Y · 3. New LT#

Exe(19, X, Y, NEW_SPRITE) — ExecSetSprite

Effect: Change the sprite number of a hex.

Arguments: 1. X · 2. Y · 3. New Sprite#

Exe(20, X, Y, AREASLOT, AREACODE) — ExecSetSlot

Effect: Set an area/hex slot value at (x,y).

Arguments: 1. X · 2. Y · 3. AreaSlot · 4. AreaCode

Exe(21, PEOPLE, PEOPLEGROUPOWNER, NEW_COMBAT_MOD) — ExecChangePeopleCombatMod

Effect: Modify combat modifiers for a people ID and group.

Arguments: 1. People# · 2. PeopleGroupOwner# · 3. New Combat Mod in %

Exe(22, MOVETYPE, PERCENTAGE_NORMAL_MOVEC) — ExecMoveTypeModifier

Effect: Set a movement-type modifier in percent.

Arguments: 1. MoveType# · 2. Percentage of normal movecost (100%=normal)

Exe(23, UNITTYPE, PERCENTAGE_NORMAL_COMBA) — ExecUnitTypeModifier

Effect: Set combat modifier for a unit type/group.

Arguments: 1. UnitType# · 2. Percentage of normal combat att/def (100%=normal)

Exe(24, RED_COMPONENT, GREEN_COMPONENT, BLUE_COMPONENT) — ExecWheaterColor

Effect: Set weather display colour components (RGB).

Arguments: 1. Red Component (0=no change, 255=full) · 2. Green Component (0=no change, 255=full) · 3. Blue Component (0=no change, 255=full)

Note: Changes the colour of the prehex sprite. But only of its core sprite not if its optional 64 border sprites.

Exe(25, X, Y, RIVER, TOO_RIVER) — ExecChangeRiver

Effect: Replace a specified river type around a hex with another river type.

Arguments: 1. X · 2. Y · 3. From River # · 4. Too River #

Exe(26, X, Y, ROAD, TOO_ROAD) — ExecChangeRoad

Effect: Replace a specified road type around a hex with another road type.

Arguments: 1. X · 2. Y · 3. From Road # · 4. Too Road #

Exe(27, X, Y, DIRECTION_0, NO_BRIDGE_0) — ExecChangeBridge

Effect: Change bridge presence along a specified edge of a hex.

Arguments: 1. X · 2. Y · 3. Direction = 0(North),1,2,3,4,5(NorthWest) · 4. -1=no bridge, >=0 = bridge

Exe(28, X, Y, REGIME_NR, AIVP_AMOUNT_0) — AI_SetAIVP

Effect: Set AI victory-point priority for a hex and regime.

Arguments: 1. X · 2. Y · 3. Regime Nr# · 4. AIVP amount. 0(none) - 999

Note: Remember that setting the name to nothing '' disables the extra tab. Also remember that the event called is supposed to generate 1 message only: no more, no less. The message contents should fit in a 1000x200 pixel area. Not setting the third parameter or to 0 results in tab#=1.

Exe(29, X, Y, LOCTYPE, PEOPLE) — ExecChangeLocationType

Effect: Create/change a location type at (x,y), with people selection and map updates.

Arguments: 1. X · 2. Y · 3. Loctype (-1=remove any location) · 4. People #

Exe(30, X, Y, SUPPLY_POINTS_TOP, OVERRULE_RECIPIENT_REGI) — ExecGiveSupply

Effect: Distribute supply points at or near selected hex according to supply rules.

Arguments: 1. X · 2. Y · 3. Supply Points to the top hq of the closest hq to x,y · 4. Overrule recipient regime with regime #

Exe(32, X, Y, CHANGES_PEOPLE_AT) — ExecChangePeople

Effect: Change the people assigned to a hex/location.

Arguments: 1. X · 2. Y · 3. Changes the people at location x,y

Exe(33, X, Y, PREDEF_USE, MULTIPLIER) — ExecGiveReinforcement

Effect: Give predefined reinforcements at a hex to a regime/HQ.

Arguments: 1. X · 2. Y · 3. Predef# to use · 4. Multiplier

Exe(34, CARD_NR, REGIME_NR) — ExecGiveActionCard

Effect: Give an action card to a regime.

Arguments: 1. Card Nr · 2. Regime Nr

Exe(35, REGIME_NR, PEOPLE_NR, RDN_LOSS, CHANCE_PER_UNIT) — ExecReduceReadiness

Effect: Apply readiness losses to units filtered by regime/people with chance.

Arguments: 1. Regime Nr (-1=all) · 2. People Nr (-1=all) · 3. %rdn loss · 4. %chance per unit

Exe(36, CARD_NR, REGIME_NR) — ExecAddHistoryActionCard

Effect: Add an action card to a regime's historical card list.

Arguments: 1. Card Nr · 2. Regime Nr

Exe(37, CARD_NR, REGIME_NR) — ExecRemoveHistoryActionCard

Effect: Remove an action card from historical list.

Arguments: 1. Card Nr · 2. Regime Nr

Exe(39, REGIME_NR, PEOPLE_NR, RDN_LOSS, LANDSCAPETYPE_NR) — ExecLTReduceReadiness

Effect: Apply readiness losses filtered by landscape type.

Arguments: 1. Regime Nr · 2. People Nr (-1=all) · 3. %rdn loss · 4. LandscapeType Nr

Exe(40, REGIME_NR, PEOPLE_NR, MOR_LOSS, LANDSCAPETYPE_NR) — ExecLTReduceMorale

Effect: Apply morale losses filtered by landscape type.

Arguments: 1. Regime Nr · 2. People Nr (-1=all) · 3. %mor loss · 4. LandscapeType Nr

Exe(41, ITEMTYPE_NR, SET_PROD_COST) — ExecSetItemCost

Effect: Set a production cost on an item type.

Arguments: 1. ItemType Nr · 2. Set Prod Cost

Exe(42, REGNR, BASEMORALE) — ExecSetBaseMorale

Effect: Set regime base morale, updating dependent morale values.

Arguments: 1. Regnr · 2. Basemorale

Note: Not only sets base morale of regime, but also of unterregimes to the same as well as reducing all morale of all units with the same percentual ammount as the basemorale lost.

Exe(43, X, Y, NEW_VP_AMOUNT) — ExecChangeVP

Effect: Set the VP value of a hex.

Arguments: 1. X · 2. Y · 3. new VP amount. 0(none) - 999

Exe(44, CARD_NR, REGIME_NR) — ExecRemoveActionCard

Effect: Remove a currently held action card from a regime.

Arguments: 1. Card Nr · 2. Regime Nr

Exe(45, REGIME_NR, CONVERVATIVE_100_NORMAL) — ExecAIConservative

Effect: Set the regime AI conservative behaviour percentage.

Arguments: 1. Regime Nr · 2. Convervative.. 100% = normal >100% more defensive. <100 more aggressive

Exe(46, REGIME_NR, ACTION_CARD_NR) — ExecPlayActionCard

Effect: Execute or play an action card on behalf of a regime.

Arguments: 1. Regime Nr · 2. Action Card Nr

Exe(49, ACTIONCARD_NR, NEW_PPCOST, PPCOST_ABSOLUTE_INCREAS, PPCOST_INCREASE_DECREAS) — ExecActionCardPPCost

Effect: Change an action card's political-point cost, with optional modifiers.

Arguments: 1. Actioncard Nr · 2. New PPCost (-1 = keep same) · 3. PPCost absolute increase or decrease (0=no change) · 4. PPCost % increase or decrease (0=no % change)

EXECS 052–100

Execs modify scenario state or produce a defined event action.

Exe(52, AREASLOT, AREACODE, SFTYPE, TOO_SFTYPE) — ExecMutateSFType

Effect: Mutate subformation types for matching troops in an area-code region.

Arguments: 1. AreaSlot · 2. AreaCode · 3. From Sftype# · 4. Too Sftype# (-1=remove)

Note: After completion sets # of changed individuals in tempvar[999].

Exe(53, AREASLOT, AREACODE, SFTYPE, TOO_SFTYPE) — ExecSingleMutateSFType

Effect: Mutate a single matching subformation type in an area region.

Arguments: 1. AreaSlot · 2. AreaCode · 3. From Sftype# · 4. Too Sftype# (-1=remove)

Note: Note: Currents turn units take precedence.

Exe(54, X, Y, SFTYPE, TOO_SFTYPE) — ExecSingleMutateSFTypeXY

Effect: Mutate a single matching subformation type at hex coordinates.

Arguments: 1. X · 2. Y · 3. From Sftype# · 4. Too Sftype# (-1=remove)

Note: Note: Currents turn units take precedence.

Exe(55, MOVETYPE, PERCENTAGE_CURRENT_AP) — ExecAntiSupplyRangeMod

Effect: Change a movement type's action/supply range.

Arguments: 1. MoveType# · 2. Percentage of current AP (100%=normal)

Exe(56, PEOPLE, VERSUS_PEOPLEGROUPOWNER, NEW_COMBAT_MOD) — ExecChangePeopleCombatModVS

Effect: Change a people/group combat modifier against another category.

Arguments: 1. People# · 2. Versus PeopleGroupOwner# · 3. New Combat Mod in %

Exe(57, X, Y, SFTYPE_GROUP, PERCENTAGE_REMOVE) — ExecRemoveTroops

Effect: Remove a percentage of troops by subformation group at a hex.

Arguments: 1. X · 2. Y · 3. SFType group · 4. Percentage to remove

Exe(58, X, Y, SPECIFIC_PEOPLE, PERCENTAGE_REMOVE) — ExecRemoveTroopsByPeople

Effect: Remove a percentage of troops belonging to a people at a hex.

Arguments: 1. X · 2. Y · 3. specific people # · 4. Percentage to remove

Exe(59, X, Y, MORALE_THEN, PERCENTAGE_REMOVE) — ExecRemoveTroopsByLowMorale

Effect: Remove troops below a morale threshold at a hex.

Arguments: 1. X · 2. Y · 3. Morale =< then · 4. Percentage to remove

Exe(61, AREASLOT, VALUE_SLOT, READINESS_LOSS, CHANCE_LOSE_THIS) — ExecReadinessLossByArea

Effect: Apply readiness loss to troops in a specified area slot/value.

Arguments: 1. Areaslot · 2. Value of slot · 3. Readiness loss · 4. Chance to lose this (x%)

Exe(62, X, Y, SFTYPEGROUP, ABSOLUTE_MODIFIER) — ExecOffensiveMod

Effect: Apply offensive combat modifier to a subformation group at a hex.

Arguments: 1. X · 2. Y · 3. SfTypeGroup# (-1=all) · 4. Absolute Modifier (0=none, -50 = -50%, +50 = +50%)

Exe(63, X, Y, SFTYPEGROUP, ABSOLUTE_MODIFIER) — ExecDefensiveMod

Effect: Apply defensive combat modifier to a subformation group at a hex.

Arguments: 1. X · 2. Y · 3. SfTypeGroup# (-1=all) · 4. Absolute Modifier (0=none, -50 = -50%, +50 = +50%)

Exe(64, X, Y, PEOPLE, ABSOLUTE_MODIFIER) — ExecOffensiveModByPeople

Effect: Apply offensive modifier to units of a specified people at a hex.

Arguments: 1. X · 2. Y · 3. People# (-1=all) · 4. Absolute Modifier (0=none, -50 = -50%, +50 = +50%)

Exe(65, X, Y, PEOPLE, ABSOLUTE_MODIFIER) — ExecDefensiveModByPeople

Effect: Apply defensive modifier to units of a specified people at a hex.

Arguments: 1. X · 2. Y · 3. People# (-1=all) · 4. Absolute Modifier (0=none, -50 = -50%, +50 = +50%)

Exe(66, REGIME, PEOPLE, SFTYPEGROUP, ABSOLUTE_MODIFIER) — ExecOffensiveModByRegime

Effect: Apply offensive modifier to matching regime/people/group troops.

Arguments: 1. Regime # (-1=all) · 2. People # (-1=all) · 3. SfTypeGroup# (-1=all) · 4. Absolute Modifier (0=none, -50 = -50%, +50 = +50%)

Exe(67, REGIME, PEOPLE, SFTYPEGROUP, ABSOLUTE_MODIFIER) — ExecDefensiveModByRegime

Effect: Apply defensive modifier to matching regime/people/group troops.

Arguments: 1. Regime # (-1=all) · 2. People # (-1=all) · 3. SfTypeGroup# (-1=all) · 4. Absolute Modifier (0=none, -50 = -50%, +50 = +50%)

Exe(72, AREASLOT, DIVIDEBY) — ExecAreaSpreadOut

Effect: Spread/degrade area-code values over the map.

Arguments: 1. AreaSlot # · 2. Divideby

Exe(73, LOCTYPE, LANDSCAPETYPE, SPRITE) — ExecSetLoctypeDestruct

Effect: Apply location destruction rules for landscape/location filters.

Arguments: 1. LocType# · 2. LandscapeType# · 3. Sprite #

Exe(74, PREDEF) — CheckPreDef

Effect: Set research cost in the specified production/people group.

Arguments: 1. Predef#

Note: Returns unit# of this predef ID#.

Exe(79, REGIME) — ExecGetReconFrom

Effect: Copy/recompute reconnaissance information for a regime.

Arguments: 1. From Regime#

Exe(80, PEOPLE, POINTS_ADD, MINIMUM_XP, MAXIMUM_XP) — ExecAddExperience

Effect: Add unit or regime experience, using the supplied selectors.

Arguments: 1. People# (-1=all) · 2. Points to add (+/-) · 3. Minimum XP · 4. Maximum XP

Exe(81) — ExecMaxResearch

Effect: Advance research up to available maximum under rules.

Arguments: None.

Note: Every Suformation thats not in a predef unit that can upgrade if its regime has the research will upgrade.

Exe(82, X, Y, ACTIONCARD) — ExecHexActionCard

Effect: Set the temporary hex-action-card state.

Arguments: 1. X · 2. Y · 3. ActionCard# (-1=no actioncard)

Note: Will place an action card on the hex in question. The card is executed immediatly upon the player conquering the hex. And will cause a popup to appear immediatly (if message used) Only if hex is conquered by combat or direct move in. So always place on locations because they cannot be conquered by ZOC. If an AI regime takes the card it cannot load a new game, select a unit or a hex.

Exe(83, HISTORIC_HQ, DEFEND_DEFINED_AREA, ATTACK_DEFINED_AREA, STANCE) — ExecSetHistoricUnitMove

Effect: Issue AI historical-move instructions for a historical unit.

Arguments: 1. Historic HQ # · 2. Defend Defined Area # · 3. Attack Defined Area # · 4. Stance (1=def,2=norm,3=att)

Exe(84, REGIME, X, Y, POWER_POINTS) — ExecChangeAIPower

Effect: Change the AI power weighting for a particular regime/hex.

Arguments: 1. Regime · 2. X · 3. Y · 4. Power Points

Note: If positive power points given it is meant to help influence the behaviour of the AI. This way you can set reinforcements to a certain regime in the area that hex x,y is in. But you can also use this function to set negative powerpoints versus a regime to simulate immediate pressure and force the AI to keep a certain reserve in the area.

Exe(90, EVENT_PICTURE_BE) — ExecLoadCampaignRoomBackground

Effect: Change campaign-room background bitmap slot.

Arguments: 1. Event Picture# to be used

Exe(95, UNIT, ABSOLUTE_MODIFIER_PTS, PEOPLE, SFTYPEGROUP) — ExecUnitRdnModify

Effect: Modify a unit's subformation readiness according to people/group selectors.

Arguments: 1. Unit # · 2. Absolute modifier pts +/- · 3. People (-1=all) · 4. SFTypeGroup (-1=all)

Note: Change stats of subformations in a specified unit.

Exe(96, UNIT, ABSOLUTE_MODIFIER_PTS, PEOPLE, SFTYPEGROUP) — ExecUnitMorModify

Effect: Modify a unit's subformation morale according to people/group selectors.

Arguments: 1. Unit # · 2. Absolute modifier pts +/- · 3. People (-1=all) · 4. SFTypeGroup (-1=all)

Note: Change stats of subformations in a specified unit.

Exe(97, UNIT, ABSOLUTE_MODIFIER_PTS, PEOPLE, SFTYPEGROUP) — ExecUnitXpModify

Effect: Modify a unit's subformation experience according to people/group selectors.

Arguments: 1. Unit # · 2. Absolute modifier pts +/- · 3. People (-1=all) · 4. SFTypeGroup (-1=all)

Note: Change stats of subformations in a specified unit.

Exe(98, UNIT, ABSOLUTE_MODIFIER_PTS, MOVETYPEGROUP, SFTYPEGROUP) — ExecUnitApModify

Effect: Modify a unit's subformation action points by movement/group selectors.

Arguments: 1. Unit # · 2. Absolute modifier pts +/- · 3. MoveTypeGroup (-1=all) · 4. SFTypeGroup (-1=all)

Note: Change stats of subformations in a specified unit.

Exe(99, UNIT, PREDEF_UNIT) — ExecUnitGiveTroops

Effect: Give troops to a unit from a predefined unit template.

Arguments: 1. Unit # · 2. Predef Unit#

Note: Add all troops in PreDef unit to this Unit.

Exe(100, UNIT, ABSOLUTE_MODIFIER_PTS, PEOPLE, SFTYPEGROUP) — ExecUnitOffensiveBonus

Effect: Modify unit offensive bonus by people/subformation group.

Arguments: 1. Unit # · 2. Absolute modifier pts +/- · 3. People (-1=all) · 4. SFTypeGroup (-1=all) (>=100 = specific sftype -100) (9001=only for artRange>0)

Note: Change stats of subformations in a specified unit. If you set modifier to 0 it resets the offmod to 0. Each point is a percentage point. If you specify a sftypegroup >=100 then you are actually specifiying a specific SFType Nr.

EXECS 101–150

Execs modify scenario state or produce a defined event action.

Exe(101, UNIT, ABSOLUTE_MODIFIER_PTS, PEOPLE, SFTYPEGROUP) — ExecUnitDefensiveBonus

Effect: Modify unit defensive bonus by people/subformation group.

Arguments: 1. Unit # · 2. Absolute modifier pts +/- · 3. People (-1=all) · 4. SFTypeGroup (-1=all) (>=100 = specific sftype -100)

Note: Change stats of subformations in a specified unit. If you set the modifier to 0 it resets the defmod to 0. Each point is a percentage point. If you specify a sftypegroup >=100 then you are actually specifiying a specific SFType Nr.

Exe(102, UNIT) — ExecUnitSelectable

Effect: Set the unit's selectable flag.

Arguments: 1. Unit #

Note: The specified unit will now be selectable in the 'select a unit' popup screen. Use this exec in a pre-event from an actioncard.

Exe(105, REGIMENR1, REGIMENR2, DIPLOMATIC_RELATION, HIDE_MESSAGE) — ExecChangeRelation

Effect: Set relationship code between two regimes.

Arguments: 1. RegimeNr1 · 2. RegimeNr2 · 3. Diplomatic Relation # · 4. Hide Message (1=yes hide)

Note: Sets the relation between regime 1 and 2 to relation 0-2. 0=war, 1=peace, 2=allied.

Exe(108, REGIME, HISTORICAL_UNIT) — ExecAddOOB

Effect: Add Order of Battle entries for a regime and historical unit.

Arguments: 1. Regime · 2. Historical Unit

Note: Sets the historical unit to the regime and also everything in the OOB of the selected historical unit.

Exe(109, REGIME, RED, GREEN, BLUE) — ExecSetRegimeColor

Effect: Set a regime's RGB map colour.

Arguments: 1. Regime · 2. Red (0-255) · 3. Green (0-255) · 4. Blue (0-255)

Note: Sets the color of the regime. And you can overwrite the Name if you want too. Leave string blank to keep name.

Exe(110, X, Y, HISTORICAL_UNIT_ID, DISPERSION_HEX) — ExecAddHistoricalUnit

Effect: Create a historical unit and deploy its associated units at a hex.

Arguments: 1. X · 2. Y · 3. Historical Unit ID · 4. Dispersion in Hex

Note: If Historical unit is a Model an instance of it will be created. If it is not it will be put on the map itself. If you set Dispersion to 0 all will go normal. Set dispersion>0 to simulate scatter effect on same sea/land type and regime type as selected for target hex. Set x=-1 to only create a new historical unit (based on specified model) and not place any unit on the map.

Exe(111, HISTORICAL_UNIT) — ExecRemoveHistoricalUnit

Effect: Remove a historical unit and its unit data.

Arguments: 1. Historical Unit

Note: All units assigned to this exact Historical Unit will be removed.

Exe(112, GAMEVAR, VALUE) — ExecSetGameVar

Effect: Assign GameSlot(index) = value directly; extra arguments are ignored.

Arguments: 1. GameVar # · 2. New Value

Exe(113, TEMPVAR, VALUE) — ExecSetTempVar

Effect: Assign TempVar(index) = value directly; extra arguments are ignored.

Arguments: 1. TempVar # · 2. New Value

Exe(114, REGIME, VARIABLE, VALUE) — ExecSetRegimeVar

Effect: Set a numbered RegimeSlot variable.

Arguments: 1. Regime # · 2. Variable # · 3. New Value

Exe(116, STRINGLIST_ID, ROW, COL, VALUE) — ExecSetStringList

Effect: Store a numeric value in a selected StringList cell. StringList storage may be textual internally, but pass numbers only in these expressions.

Arguments: 1. Stringlist ID # · 2. Row · 3. Col · 4. New Value

Numeric use only: pass a number as the cell value. Text values belong to other Event Editor workflows.

Exe(118, X, Y, REGNR, PTS) — ExecSetRecon

Effect: Change reconnaissance at a hex/region for a regime.

Arguments: 1. X · 2. Y · 3. RegNr · 4. Pts

Note: Sets recon for this regime only on this hex only.

Exe(119, UNIT, COMBATMOD, MORALEMOD, STAFFPTS) — ExecChangeCommanderSkill

Effect: Modify a commander's skill.

Arguments: 1. Unit# · 2. CombatMod +/- · 3. MoraleMod +/- · 4. StaffPts +/-

Note: Inrease or diminish the officers stats.

Exe(120, UNIT, TEMPVAR, VALUE, USE_HISTORICAL_UNIT) — ExecChangeHisTempValue

Effect: Modify commander temporary variables.

Arguments: 1. Unit# (-1= use hist.unit#) · 2. TempVar#(1-5), 6=aiZoneListNr · 3. New Value · 4. Use Historical Unit#

Note: Keep in mind you can also use hisvars to store data. of which you have unlimited quantity. For aiZoneListNr only number above >0 are considered valid.

Exe(123, UNIT, STRINGLIST_CARDLIST_ID) — ExecCommanderAddDeckCardUsingStringList

Effect: Add card to a commander's deck.

Arguments: 1. Unit# · 2. StringList CardList ID#

Note: Picks a weighted chance card from the stringlist in question. will not take doubles. will add text to description.

Exe(124, UNIT, STRINGLIST_EVENTLIST_ID) — ExecCommanderAddAutoEventUsingStringList

Effect: Add automatic event to commander.

Arguments: 1. Unit# · 2. StringList EventList ID#

Note: Picks a weighted auto-event from the stringlist in question. will not take doubles. will add text to description.

Exe(125, ROWS, COLS) — ExecAddStringList

Effect: Add a StringList object.

Arguments: 1. Rows · 2. Cols

Note: Adds a whole new stringlist object to the game data. Use the CheckStringListLastID to get its ID, which you need to change or read anything from it.

Exe(126, STRINGLIST_ID, EXTRA_ROWS, EXTRA_COLS) — ExecAddStringListCells

Effect: Add rows/columns/cells to a StringList.

Arguments: 1. StringList ID · 2. Extra Rows · 3. Extra Cols

Note: If you specify 0 will NOT add anything. specify 1 for adding one row or col. You can add rows and cols at the same time. Rows are added at the bottom, Cols on the right.

Exe(127, STRINGLIST_ID, ROW_NUMBER) — ExecRemoveStringListRow

Effect: Remove a StringList row.

Arguments: 1. StringList ID · 2. Row Number

Exe(128, STRINGLIST_ID, COL_NUMBER) — ExecRemoveStringListCol

Effect: Remove a StringList column.

Arguments: 1. StringList ID · 2. Col Number

Exe(129, REGIME, X, Y, DEFENSIVE_BONUS) — ExecChangeAIDefensive

Effect: Change AI defensive power/priority for selected regime and hex.

Arguments: 1. Regime · 2. X · 3. Y · 4. Defensive Bonus %

Note: >0 is a percentual defensive bonus for defending this area for this regime. NEW AI ONLY.

Exe(130, UNIT, XP_GIVE_TAKE) — ExecGiveCommanderXP

Effect: Give experience points to a commander.

Arguments: 1. Unit# · 2. XP to give or take +/-

Exe(131, REGIME, OVERRULE_OFFICER_POOL) — ExecCreateCommander

Effect: Create a commander entry.

Arguments: 1. Regime # · 2. Overrule Officer Pool StringList ID (-1=dont)

Exe(132, UNIT, NEW_X, NEW_Y) — ExecMoveUnit

Effect: Move a unit to a new hex, handling ownership/combat contingencies.

Arguments: 1. Unit # · 2. New X · 3. New Y

Note: Keep in mind combat can occur immediatly after the move. Rebel Battle Mode will be used.

Exe(133, UNIT, NEW_REGIME) — ExecChangeUnitRegime

Effect: Change a unit's owning regime, with map/combat updates.

Arguments: 1. Unit # · 2. New Regime#

Note: Keep in mind combat can occur immediatly after the move. Rebel Battle Mode will be used.

Exe(134, LANDSCAPE, MOVEMENTTYPE, AP_COST) — ExecChangeLandscapeMoveCost

Effect: Change landscape movement-cost modifier.

Arguments: 1. Landscape # · 2. MovementType # · 3. AP Cost

Exe(136, SFTYPE, SFTYPEGROUP, MODIFIER) — ExecModifyAirOverrule

Effect: Modify air overrule fields for a subformation type.

Arguments: 1. SFType # (-1=all) · 2. SFTypeGroup # (-1=all) · 3. Modifier (100=same, 50=half, 200=double) #

Exe(137, SFTYPE, SFTYPEGROUP, MODIFIER) — ExecModifyEntrenchmentSpeed

Effect: Modify entrenchment settings.

Arguments: 1. SFType # (-1=all) · 2. SFTypeGroup # (-1=all) · 3. Modifier (100=same, 50=half, 200=double) #

Exe(138, SFTYPE, SFTYPEGROUP, MODIFIER, DC4_ONLY_SPECIAL) — ExecModifySupplyStock

Effect: Modify supply/fuel store sizes; supports SF-type and unit-group selectors.

Arguments: 1. SFType # (-1=all) · 2. SFTypeGroup # (-1=all) · 3. Modifier (100=same, 50=half, 200=double) # · 4. For dc4+ only. Special Mode. <1=dont use. 1=Supply, 2=Fuel

Note: Modifies supply and stockpile stats.

Exe(139, SFTYPE, SFTYPEGROUP, VERSUS_SFTYPEGROUP, MODIFIER) — ExecModifyAttackScore

Effect: Scales attack values against a target subformation group by the specified percentage.

Arguments: 1. SFType # (-1=all) · 2. SFTypeGroup # (-1=all) · 3. Versus SFTypeGroup # (-1=all) · 4. Modifier (100=no change; 50=half; 200=double)

Exe(140, REGIME, SPECIFIC_UNIT, MODIFIER, MODIFIER_ABSOLUTE) — ExecModifyUnitSupply

Effect: Modify current unit supply points.

Arguments: 1. Regime (-1=none) · 2. Specific Unit (-1=none) · 3. Modifier % (100=keep same, 50=half, 200=double) · 4. Modifier absolute (0=nill, -x , +x)

Note: Increases or decreases supplies of the unit. cannot go over max supplystore.

Exe(141, REGIME) — ExecRemoveUnits

Effect: Remove a unit from game data.

Arguments: 1. Regime #

Exe(143, X, Y, REINFORCEMENT_GROUP, PERCENTAGE_REMOVE) — ExecRemoveTroopsByReinforcementGroup

Effect: Remove a percentage of troops by reinforcement type on a hex.

Arguments: 1. X (-1 = from unit) · 2. Y (or unit #) · 3. Reinforcement group · 4. Percentage to remove

Exe(144, UNIT, HISTORICAL_UNIT) — ExecChangeCommander

Effect: Assign/change a commander for a historical unit.

Arguments: 1. Unit# · 2. Historical Unit#

Exe(145, SOURCE_X, SOURCE_Y, AREASLOT, REGIME) — ExecSetSupplyMatrix

Effect: Populate an area-slot matrix with predicted supply/access from a hex.

Arguments: 1. Source X · 2. Source Y · 3. AreaSlot # · 4. Regime #

Note: Sets movecost value on each hex. if not in supply value will be set to 9999. It will not overwrite a lower value with a higher value. So make sure you set the areaslot matrix to 9999 before you call. this is done to allow multiple scenarios for making the matrix.

Exe(146, AREASLOT, VALUE) — ExecClearMatrix

Effect: Clear a matrix/area slot to selected value.

Arguments: 1. AreaSlot # · 2. Value

Note: Sets the specified slot for all hexes to specified value.

Exe(147, FORREGIME, UBERREGIME) — SetUberRegime

Effect: Set the overlord (UberRegime) of a regime.

Arguments: 1. ForRegime · 2. UberRegime (-1=none)

Exe(148, X, Y) — SetCardXY

Effect: Store action-card selected X/Y coordinates.

Arguments: 1. X · 2. Y

Note: If you are calling a card who needs a hex input. set it with this exec.

Exe(149, REINFORCEMENT_GROUP, LOSS) — ExecRemoveTroopsForNextHistorical

Effect: Record a historical reinforcement removal request for the event.

Arguments: 1. Reinforcement group · 2. Loss %

Note: For each event you have to set the loss levels. These values are reset per event. If set any unit added with AddHistorical will suffer these losses.

Exe(150, HISTORICAL_UNIT, NEW_COUNTER_VALUE) — ExecSetHistoricalCounter

Effect: Change the historical-unit count/counter.

Arguments: 1. Historical Unit · 2. New Counter Value

EXECS 151–200

Execs modify scenario state or produce a defined event action.

Exe(151, HISTORICAL_UNIT, NEW_UNIT_HQ, HISTORICAL_UNIT_HQ) — ExecSetHistoricalUnitHQ

Effect: Assign HQ relationships for historical-unit data.

Arguments: 1. Historical Unit · 2. New unit # HQ · 3. Historical Unit # HQ

Note: Sets all units that belong to this historical unit to the unit or historical unit HQ # specified.. Historical <= 0 means unit is used.

Exe(152, X, Y, REGIME, POINTS) — ExecAddAntiSupply

Effect: Apply anti-supply points at (x,y) against a regime.

Arguments: 1. X · 2. Y · 3. Regime · 4. Points

Note: Must be used in early turn check in order to be used after regular setting of AS and before Supply system.

Exe(153, DAY, MONTH, YEAR) — ExecChangeDate

Effect: Set start date from supplied day/month/year, under validity checks.

Arguments: 1. Day · 2. Month · 3. Year

Exe(155, REGIME, STANCE) — ExecAIStance

Effect: Set AI stance field of specified regime.

Arguments: 1. Regime · 2. Stance (0=no overrule, 1=att, 2=def, 3=meeting)

Note: If 0 the regime with most powerpoints is considered the attacker. If 3 too, but if 3 then this regime also gets the meeting flag.

Exe(156, LANDSCAPETYPE, PERCENTAGE) — ExecReduceLandInitialEntrench

Effect: Reduce initial entrenchment by landscape/type filters.

Arguments: 1. LandscapeType (-1=all) · 2. Percentage (100=same. 50=less >100=more)

Exe(157, SPECIALTYPE, SPECIALSPRITE, X, Y) — ExecSetSpecial

Effect: Set a hex's special type and special sprite.

Arguments: 1. SpecialType · 2. SpecialSprite · 3. X · 4. Y

Note: Either Type is a LanscapeType slot with corresponding Sprite or it is -1 and a smallgfx slot will be used.

Exe(158, SFTYPE, PERCENTAGE) — ExecReduceAntiStructural

Effect: Reduce anti-structural properties.

Arguments: 1. SFType (-1=all) · 2. Percentage (100=same. 50=less >100=more)

Exe(159, SFTYPE, PEOPLETYPE, REMOVE_X, REMOVE_BETWEEN_0) — ExecReduceTroops

Effect: Reduce troop quantities under specified filters.

Arguments: 1. SFType (-1=all) · 2. PeopleType (-1=all) · 3. Remove x% · 4. + remove between 0-y% random

Exe(161, SFTYPE, TARGETGROUP, ATTACK_SCORE, DEFEND_SCORE) — ExecSFTypePower

Effect: Set a subformation type's attacking/defending combat power, by mode.

Arguments: 1. SFType · 2. Targetgroup# (-1=all) · 3. Attack score · 4. Defend score

Exe(164, SFTYPE, TARGETGROUP, ATTACK_HP, DEFEND_HP) — ExecSFTypeHP

Effect: Alias of Exec(160): modifies subformation hit points by mode.

Arguments: 1. SFType · 2. Targetgroup# (-1=all) · 3. Attack HP · 4. Defend HP

Exe(165, SFTYPE, TYPE) — ExecSFTypeMoveType

Effect: Set movement type for a subformation type.

Arguments: 1. SFType# · 2. Type

Exe(166, SFTYPE, REDUX) — ExecSFTypeMoveRedux

Effect: Set movement-reduction value for a subformation type.

Arguments: 1. SFType# · 2. Redux

Exe(170, SFTYPE, FORMOVE, FORATTACK) — ExecSFTypeFuelUse

Effect: Set subformation fuel used for movement and attack.

Arguments: 1. SFType# · 2. ForMove (10ap) · 3. ForAttack (combatround)

Exe(171, REGIME, PEOPLE, OFFICERPOOL) — ExecSetRegimePeople

Effect: Assign a regime's people ID.

Arguments: 1. Regime# · 2. People # (or -1=to leave as is) · 3. officerpool # (or-1=to leave as is)

Exe(172, UNIT, HQ) — ExecSetHQ

Effect: Set the HQ of a particular unit.

Arguments: 1. Unit · 2. HQ

Exe(173, SLOT, OPERATION, DISALLOW_OVER_SEA) — ExecSetMatrix

Effect: Compute/fill a matrix/area-slot using a selected operation and sea rule.

Arguments: 1. Slot · 2. Operation · 3. Disallow over sea (0=no, 1=yes)

Note: You cannot set operation yet. For now it just subtracts 1 of the value and copies if higher then neighbour. Thus make sure to make a clearmatrix(0) and then set coordinates you want to spread out.

Exe(174, X, Y, OWNER) — ExecSetHexOwner

Effect: Change hex owner to the given regime.

Arguments: 1. X · 2. Y · 3. Owner #

Exe(175, X, Y, POWER_ABOVE, MULTI) — ExecRewardPowerOn

Effect: Set temporary reward power/multiplier fields on a hex.

Arguments: 1. X · 2. Y · 3. Power Above · 4. Multi

Note: If AI can get more then power above on X,Y coordinate. all pts above this treshold will be multiplied by multi and added to score. This gives AI incentive to get a certain build up here. so it must be able to hold it to get it.. This is only used in Init AI so if you use uberregime only the uberregime needs this setting. This setting is wiped clean after every AI has played. You thus need to set this in early turn event.

Exe(176, UNIT) — ExecRemoveunit

Effect: Remove a specified unit without general selector loops.

Arguments: 1. Unit #

Exe(179, STRINGLISTID, COLUMN_SORT, MODE) — ExecSortStringListID

Effect: Sort a StringList by specified column and order mode.

Arguments: 1. StringListID# · 2. Column# to sort on · 3. 0=asc, 1=desc

Note: Remember first column is column 0. ASCENDING: highest at top. DESCENDING: lowest at top.

Exe(180, UNIT, ENTRENCHMENT_PTS) — ExecChangeUnitEntrenchment

Effect: Change a unit's entrenchment value.

Arguments: 1. Unit# · 2. -/+ entrenchment pts

Note: Will auto cut at 0 or maximum entrench for sftype in hex.

Exe(181, SFTYPE, VAR, VALUE) — ExecSetSFTypeVar

Effect: Set numbered SFTypeVar for a subformation type.

Arguments: 1. SFType · 2. Var# · 3. Value

Exe(182, HISTORIC_HQ_UNIT, GROUP_HQ_UNIT) — ExecSetHistoricUnitToGroup

Effect: Reassign a historical unit to an HQ group.

Arguments: 1. Historic HQ/Unit/Div # · 2. To Group with HQ/Unit/Div# in it

Note: This should be called in an 'AI Init Check' since it re-orders strategic groups (after makegroups) and before going into strategic analysis (findbeststrategy).

Exe(183, HISTORICAL_UNIT, NEW_X, NEW_Y, OPTIONAL_HISTORICAL_HQ) — ExecMoveHistoricalUnit

Effect: Move the map unit belonging to the specified historical unit.

Arguments: 1. Historical Unit # · 2. New X · 3. New Y · 4. Optional Historical HQ #

Note: If you specify >0 for optional HQ all units attached to the HQ will be moved. you can specify -1 for historical unit if you specify hq.

Exe(186, MODE, DAYS_PER_ROUND) — ExecDateMode

Effect: Set the game's date progression mode.

Arguments: 1. 0=rounds, 1=dates · 2. days per round

Exe(187, UNIT, CHANGE_PEOPLE, TOO_PEOPLE) — ExecUnitPeopleModify

Effect: Change a unit's people assignment from one type to another.

Arguments: 1. Unit # · 2. Change People# (-1=all) · 3. Too People#

Exe(190, REGIME, COMBAT_BONUS, MOVEMENT_BONUS) — ExecAICombatMoveMod

Effect: Set AI combat or movement difficulty/help modifiers for a regime.

Arguments: 1. Regime # · 2. Combat Bonus · 3. Movement Bonus

Note: Combat bonus can be everything between 0-999%, movement bonus between 0-100%. If you set -1 for combat or move the bonus is not changed.

Exe(191, SFTYPE, AI_ROLE, NEW_SCORE) — ExecAIChangeRoleScore

Effect: Change AI role score for an SF type and role index.

Arguments: 1. SFType # · 2. AI Role # (0-49) · 3. New score

Note: Allows you to change AI role scores ingame so to influence AI production.

Exe(193) — ExecAIProductionResourceComplient

Effect: Trigger AI resource-compliance calculation.

Arguments: None.

Note: Will call the AI function that makes sure that production is research complient. If rulevar(875) is active this is called by AI after setting production. However if you script the production with events you might want to call this function add the end of that scripting in some scenarios or give the AI unlimited rescenarios instead.

Exe(194, REGIMENR1, REGIMENR2, DIPLOMATIC_OFFER) — ExecChangeDipOffer

Effect: Set diplomatic offer from one regime toward another.

Arguments: 1. RegimeNr1 · 2. RegimeNr2 · 3. Diplomatic Offer # (0=none or 1=offer)

Note: Does not give message. You have to that manually. Sets the offer mode to either 0= cancel offer or 1=give offer.

Exe(198) — ExecClearAllLabels

Effect: Remove all labels (or selected label content) from the map/hex.

Arguments: None.

Exe(199, PEOPLE, RANDOM_PERCENTAGE, SFTYPE, TOO_SFTYPE) — ExecMutateRandomSFType

Effect: Randomly mutate matching subformation types for a people.

Arguments: 1. People (-1=all) · 2. Random percentage · 3. From Sftype# · 4. Too Sftype# (-1=remove)

Note: The random percentage determines ammount of sftype to mutate. After completion sets # of changed individuals in tempvar[999].

Exe(200, SFTYPE_GROUP, PEOPLE, REINFORCEMENT_TYPE, PERCENTAGE_REMOVE) — ExecRemoveTroopsGlobal

Effect: Remove troop percentages globally using group/people/reinforcement filters.

Arguments: 1. SFType group (-1=all) · 2. People (-1=all) · 3. Reinforcement Type (-1=all) · 4. Percentage to remove

Note: Remove percentage is not exact in this function... its the chance for each individual that it will be removed. If it is set to 999 empty HQ units will also be removed. If not only empty normal units will be removed.

EXECS 201–250

Execs modify scenario state or produce a defined event action.

Exe(201, SFTYPE, PEOPLE, EXPERIENCE, MULTIPLIER) — ExecGiveReinforcement2

Effect: Give reinforcements with explicit SF type/people/XP/multiplier settings.

Arguments: 1. SFType# · 2. People# · 3. Experience · 4. Multiplier

Note: Uses SetCardXY for coordinate to drop troops input.

Exe(202, SOURCE_X, SOURCE_Y, AREASLOT, MOVETYPE) — ExecSetMoveMatrix

Effect: Create movement prediction matrix starting from hex coordinates.

Arguments: 1. Source X · 2. Source Y · 3. AreaSlot # · 4. MoveType #

Note: Sets movecost value on each hex for specified movetype. if not reachable will be set to 9999. It will not overwrite a lower value with a higher value. So make sure you set the areaslot matrix to 9999 before you call. this is done to allow multiple scenarios for making the matrix. The matrix is calculated for the regime that owns the hex.

Exe(204, HIS_UNIT_ID, TYPE) — ExecRemoveHisVar

Effect: Remove a historical-unit variable.

Arguments: 1. His.Unit ID# · 2. Type #

Exe(205, UNIT, HISTORICAL_UNIT, DECKCARD, PERCENTAGE_CHANCE) — ExecCommanderAddDeckCard2

Effect: Add card to commander deck using second-generation selector logic.

Arguments: 1. Unit# (or -1) · 2. Historical Unit# (or -1) · 3. DeckCard# · 4. Percentage Chance % (or -1,-2,-3,-4,-5,-6)

Note: Either specify a normal unit# or a historical unit#. If instead of percentage you specify -1 your remove specified card if present, if you specifify -2 you remove all deck cards. if you specify -3 you'll remove all hand cards and all deck cards. If you specify -4 you'll only remove the hand card and not the deck card. If specify -5 you will not add any card but if the card is present set its chance to 0%. If -6 then you will also not add a card but if card is present set its chance to 100%.

Exe(206, UNIT, AUTOEVENT, PERCENTAGE_CHANCE, HIS_UNIT_ID) — ExecCommanderAddAutoEvent

Effect: Add commander auto-event using second-generation selector logic.

Arguments: 1. Unit# · 2. AutoEvent# · 3. Percentage Chance % (-1=delete if present) · 4. His.Unit ID overrule#

Exe(207, X, Y, REGNR, RECON_POINTS) — ExecSetReconUnit

Effect: Update reconnaissance using second-generation rules.

Arguments: 1. X · 2. Y · 3. RegNr · 4. Recon points

Note: Sets recon for this regime only on this hex and all its neighbours up to range. Uses the ammount of recon points the specified unit# has.

Exe(208, HISTORICAL_UNIT_ID, POOL_1YES_0NO) — ExecCommanderPool

Effect: Adds or removes a historical commander from the officer pool (1=yes, 0=no).

Arguments: 1. Historical unit ID# · 2. In Pool? 1=yes, 0=no

Exe(209, HISTORICAL_UNIT, CLEAR_HANDCARDS, CLEAR_DECKCARDS, CLR_EVENTS_STAFFSIZ) — ExecClearCommander

Effect: Clear assigned commander information.

Arguments: 1. Historical Unit# (or -1) · 2. Clear HandCards (1=yes) · 3. Clear DeckCards (1=yes) · 4. Clr Events,StaffSiz,Mormod,combatmod (1=yes)

Note: Allows you to clean all cards & events associated with a historical unit.

Exe(210, HISTORICAL_UNIT, COMBATMOD, MORALEMOD, STAFFPTS) — ExecChangeCommanderSkill2

Effect: Modify commander skill using second-generation logic.

Arguments: 1. Historical Unit# · 2. CombatMod · 3. MoraleMod · 4. StaffPts

Note: Same as other function. But here the historical unit# instead of the unit# is targeted. And the setting is absolute not +/-.

Exe(211, UNIT, MODEL_ID, CHANGE_NAME_IF) — ExecSwitchUnitModel

Effect: Switch a unit's historical model / recreate according to model ID.

Arguments: 1. Unit# · 2. To Model ID# · 3. Change name as if new unit 0=no ,1=yes

Note: This function will attempt to change the whole historical unit# of unit# to a new model and will implement the changes immediately. A new name will be also be given if specified so.

Exe(212, STRINGLIST_ID) — ExecClearStringList

Effect: Clear rows/cells of a StringList.

Arguments: 1. Stringlist ID #

Exe(213, PEOPLE, UNDER_PEOPLE_GROUP, NEW_MORALE, RESET_ALL_TROOPS) — ExecSetPeopleMorale

Effect: Set morale of a people/group and optionally reset existing units.

Arguments: 1. People # · 2. Under People Group# · 3. New Morale · 4. Reset all troops 0=no,1=yes

Note: Reset=1 sets all morale to regime base morale * people morale as if in start of scenario.

Exe(214, HISTORICAL_UNIT_ID, HEX_TARGET_X, HEX_TARGET_Y, ATTACK_TOWARDS) — AI_SetTacticalScript

Effect: Set a tactical script for a historical unit at a target hex/direction.

Arguments: 1. Historical Unit ID# · 2. Hex Target X · 3. Hex Target Y · 4. Attack Towards (1-6) -1=no attack

Exe(216, UNIT, LANDCAP_PTS, NAVYCAP_PTS, AIRCAP_PTS) — ExecSetUnitCapacityPoints

Effect: Adjusts a unit's transport capacities: land, navy and air; negative results are limited to zero.

Arguments: 1. Unit# · 2. LandCap Pts +/- · 3. NavyCap Pts +/- · 4. AirCap Pts +/-

Exe(217, SFTYPE_GROUP, HISTORICAL_UNIT, REINFORCEMENT_TYPE, PERCENTAGE_REMOVE) — ExecRemoveTroopsUnderHistorical

Effect: Remove a percentage of troops under a historical unit with optional filters.

Arguments: 1. SFType group (-1=all) · 2. Historical Unit · 3. Reinforcement Type (-1=all) · 4. Percentage to remove

Note: Remove percentage is not exact in this function... its the chance for each individual that it will be removed. Compared to its namesake exec this one only affects units under command of the historical unit (HQ!) specified. Only units directly under the historical unit are affected and HQs are always excluded. (if you want to set the HQs you'll have to do it manually).

Exe(221, HISTORICAL_UNIT, PERCENTAGE_SUPPLY_LEAVE, NEW_SUPPLY_CONSUMPTION) — ExecRemoveSupplyUnderHistorical

Effect: Reduce supplies held by units under a historical command.

Arguments: 1. Historical Unit · 2. Percentage of supply to leave · 3. New Supply Consumption % (normal is 100%)

Note: this one only affects units under command of the historical unit (HQ!) specified. Only units directly under the historical unit are affected and HQs are always excluded. (if you want to set the HQs you'll have to do it manually).

Exe(222, REGIME, PERCENTAGE_INTERCEPT) — ExecSetInterceptRdnStop

Effect: Set regime interception percentage.

Arguments: 1. For Regime (-1=all) · 2. Percentage for intercept

Note: Only affects units with air units in them. percentage example: 75 means only at 75% rdn.

Exe(223, ID, MAXRECRUIT) — ExecSetHistoricalUnitMaxRecruit

Effect: Set maximum recruit value for a historical unit.

Arguments: 1. ID# · 2. MaxRecruit#(-1=unl,0=none)

Note: A function that allows you to change the max recruit.

Exe(224, ID, TYPE) — ExecSetHistoricalUnitType

Effect: Set type code of a historical unit.

Arguments: 1. ID# · 2. Type#(5=corps,6=a,7=ag,8=high)

Note: A function that allows you to change the type. this is usefull to make 1st panzer army immobile and the sov front too.

Exe(225, ADD_X_DAY, ADD_X_MONTH, ADD_X_YEAR) — ExecChangeDateAdd

Effect: Change the date with an alternative date-setting operation.

Arguments: 1. add x Day · 2. add x Month · 3. add x Year

Exe(227, REGIME, STRINGLIST_ID) — ExecRemoveActiveOfficersFromStringlist

Effect: Remove already-active officers from an officer StringList.

Arguments: 1. regime# · 2. stringlist ID

Note: Looks up the officer historical ID in column 0 and sets availability to 0 in column 1. Active officers are officers that are in the pool or on the map in a unit.

Exe(228, HISTORICALID, RANDOM_PERCENTAGE, SFTYPE, TOO_SFTYPE) — ExecMutateRandomSFTypeUnderHis

Effect: Randomly mutate subformations under a specified historical unit.

Arguments: 1. HistoricalID · 2. Random percentage · 3. From Sftype# · 4. Too Sftype# (-1=remove)

Note: The random percentage determines ammount of sftype to mutate. After completion sets # of changed individuals in tempvar[999].

Exe(229, PEOPLE, STRINGLIST_ID) — ExecRemovePeopleOfficersFromStringlist

Effect: Filter officers of selected people out of an officer StringList.

Arguments: 1. people# · 2. stringlist ID

Note: Looks up the officer historical ID in column 0 and sets availability to 0 in column 1 if the officer has specified people#.

Exe(230, REGIME, PEOPLE, VARTYPE, VARQTY) — ExecRemoveFromOfficerPool

Effect: Remove officers from a commander pool by people/variable matching.

Arguments: 1. regime# (-1=all) · 2. people# (-1=all) · 3. VarType (-1=ignore condition) · 4. =VarQty

Note: Removes any officers from the officerpool of specified regime who has the specified characeristics. You can deted vartype = varqty to tailormake catch a specific officer.

Exe(231, UNDER_HISTORICAL_UNIT, MODIFIER) — ExecModifyStockpiles

Effect: Set supply stockpile or stockpile modifiers below specified historical HQ.

Arguments: 1. Under Historical Unit# · 2. Modifier (100=same, 50=half, 200=double) #

Note: The function will cap at maximum stockpiles. So you can specify 999 or more modifier to give unit max. and 0 to set stockpile to nothing.

Exe(232, REGIME_NR, FERRY_EFF) — ExecSetRegimeFerryEff

Effect: Set a regime's ferry efficiency field.

Arguments: 1. Regime nr · 2. Ferry eff (0=none, 100=full)

Exe(234, SFTYPE, MODE, VERSUS_SFTYPEGROUP, NEW_VALUE_BE) — ExecModifyAttackScore2

Effect: Change attack-score fields using second-generation selector.

Arguments: 1. SFType # (-1=all) · 2. 1=offensive, 2=defensive · 3. Versus SFTypeGroup # (-1=all) · 4. New value to be multiplicated with current value and divided by 100

Note: 1=Modifies AttackArt,AttackPower 2=modifies: AttackPowerDef.

Exe(235, SFTYPE, VERSUS_SFTYPEGROUP, NEW_VALUE_BE) — ExecModifyHitpoints2

Effect: Modify hit points using second-generation selector.

Arguments: 1. SFType # (-1=all) · 2. Versus SFTypeGroup # (-1=all) · 3. New value to be multiplicated with current value and divided by 100

Exe(236, SFTYPE, MODE, VERSUS_SFTYPEGROUP, NEW_VALUE_BE) — ExecModifyAttackScore3

Effect: Change attack-score fields using third-generation selector.

Arguments: 1. SFType # (-1=all) · 2. 1=art attack score · 3. Versus SFTypeGroup # (-1=all) · 4. New value to be multiplicated with current value and divided by 100

Note: 1=Modifies AttackArt.

Exe(241, HIS_UNIT_ID, TYPE, VALUE, SMALLGFX) — ExecSetHisVar2

Effect: Set historical-unit variable using alternative mode.

Arguments: 1. His.Unit ID# · 2. Type # · 3. Value · 4. SmallGfx (-1= no smallgfx)

Note: If a smallGfx is specified the value and the smallgfx will be displayed for officers.

Exe(243, UNIT_NR) — ExecUnitOnTop

Effect: Move a unit to top of stack/selection order.

Arguments: 1. Unit nr#

Note: Makes sure this unit nr is on top of the stack in its hex.

Exe(246, UNIT, RETREAT, SUPPLY, REPLACEMENT) — ExecSetUnitStandingOrders

Effect: Set a unit's standing orders.

Arguments: 1. Unit# · 2. Retreat% · 3. Supply% · 4. Replacement%

Note: Give unit slot number and percentages. Allows you to set a number of standing order % to a unit. If you specify -1 then the setting is not changed. Only 0-100 values are thus accepted. Retreat100% = fight till last Retreat0%=immediate retreat.

Exe(247, SPECIAL_TYPE, SPRITE) — ExecClearSpecialTypesOnMap

Effect: Clear map hex special types.

Arguments: 1. Which special type # (-1=all) · 2. Which sprite # (-1=all)

Note: Remember that special types are just landscape types under another name.

Exe(249, CARD_SLOT, QUICK_MODE, SMALL_GRAPHIC_ORDER) — ExecSetCardQuickButton

Effect: Configure an action card's quick button.

Arguments: 1. Card Slot# · 2. Quick Mode (0=no,1=yes,2=yes+hide) · 3. Small graphic on order button

Note: If mode>=1 then an order button will be generated if the player/ selected hex/ selected unit confirms to the (pre-event) conditions of playing a card in general/ on hex/ on unit. And enough PP must be available. You must specifiy a small graphic slot # as well for the order button.

EXECS 251–298

Execs modify scenario state or produce a defined event action.

Exe(253, ACTIONCARD_NR, NEW_CATEGORY_NUMBER) — ExecActionCardCategory

Effect: Set category ID on an action card.

Arguments: 1. Actioncard Nr · 2. New category number

Note: For changing the category numner of the card.

Exe(254, CARD_NR) — ExecDeleteActionCard

Effect: Delete an action card.

Arguments: 1. Card Nr

Note: *USE WITH CAUTION* This exec actually really deletes the card. Use ExecRemoveActionCard to keep the card but remove it from a regime hand. Deleting cards will change numbers it might cause events to break.

Exe(255, CARD_NR, EVENT_PICTURE_NR) — ExecSetActionCardEventPicture

Effect: Change an action card's event-picture ID.

Arguments: 1. Card Nr · 2. Event Picture Nr

Note: A simple replace picture of the card.

Exe(256, SFTYPE, PERCENTAGE) — ExecChanceOnDeathIfMakesHit

Effect: Set chance of death after making a hit, from percent.

Arguments: 1. SFType# · 2. Percentage%

Note: 0% is will never die due to making a hit on enemy. 1=1% chance, etc.

Exe(258, X, Y, SUPPLY_POINTS_BE, AP_MODIFIER_100NORMAL) — ExecGiveSupplyImmediateDisperse

Effect: Disperse supply/fuel along accessible hexes using a movement/supply range.

Arguments: 1. X · 2. Y · 3. Supply Points to be shared by all units · 4. AP modifier %. 100=normal. 50=only half range. 200=double range.

Exe(259, ROADTYPE, MOVEMENTTYPE, AP_COST) — ExecChangeRoadTypeMoveCost

Effect: Set movement cost for a specified road type.

Arguments: 1. RoadType # · 2. MovementType # · 3. AP Cost

Exe(279, HISTORICAL_UNIT_ID, NEW_SMALL_GFX) — ExecSetHistoricalSmallGfx

Effect: Set small graphic ID of historical unit.

Arguments: 1. Historical Unit ID · 2. New Small Gfx slot#

Exe(291, HISTORICAL_UNIT_ID, RED_MODIFIER, GREEN_MODIFIER, BLUE_MODIFIER) — ExecSetHistoricalColor

Effect: Set map colour of a historical unit.

Arguments: 1. Historical Unit ID · 2. Red modifier · 3. Green modifier · 4. Blue modifier

Note: Remember you cannot set the color using Historical unit colors, only the modifier to the regime or people color of the counter. For example -50 or +100.

Exe(298, ID, TOO_ID, NUMBER_COLUMNS, MODE_1CLEAR_COPY) — ExecCopyStringlist

Effect: Performs the specified scenario action.

Arguments: 1. From ID · 2. Too ID · 3. Number of columns · 4. Mode 1=clear&copy, 2=add. columns can be set to 99 or something very high to include all

EXECS 306–341

Execs modify scenario state or produce a defined event action.

Exe(306, LOCATION_SLOT, HQ_UNIT_SLOT) — ExecChangeLocationHQ

Effect: Assign a location's HQ unit slot.

Arguments: 1. Location Slot# · 2. HQ Unit Slot#

Exe(307, HISTORICAL_UNIT_ID, REGIME_SLOT) — ExecSetHistoricalRegime

Effect: Assign a historical unit's owning regime.

Arguments: 1. Historical Unit ID · 2. Regime slot#

Exe(336, SFTYPE, TARGETGROUP, FAVORITE_SCORE_REGULAR, FAVORITE_SCORE_ART) — ExecSFTypeFavorite

Effect: Set preferred target ratings of a subformation type by target group.

Arguments: 1. SFType · 2. Targetgroup# (-1=all) · 3. Favorite score for regular attack · 4. Favorite score for art attack

Exe(337, SFTYPE, TARGETGROUP, ART_POWER) — ExecSFTypeArtPower

Effect: Set artillery attack-power rating of a subformation type by target group.

Arguments: 1. SFType · 2. Targetgroup# (-1=all) · 3. Art Power

Exe(338, SFTYPE, STAT_NUMBER, VALUE) — ExecSFTypeStat

Effect: Set SF-type statistics by numeric property selector (corresponds to Chk 253).

Arguments: 1. SFType · 2. Stat Number · 3. New Value

Note: Stat Numbers: 1=weight, 2=carrycap, 3=canDoParadrop(0/1), 4=EP, 5=BlowBridgPts, 6=BasicSupNeed, 7=SupCarry, 8=ReconPts, 9=HidePts, 10=AntiSupplyPts, 11=AntiSupplyRange, 12=AnitSupplySea, 13=InitiativeAtt, 14=InitiativeDef, 15=Attacks, 16=StackPts, 17=RearArea(0/1), 18=ArtRange, 19=Fav.Target tries, 20=AA-range, 21=HitKill%, 22=HitRetreat%, 23=ArtilleryModSfTypeNr, 24=maxAttacked, 25=moveRedux.

Exe(339, SFTYPE, LANDSCAPETYPE, OFF_MOD_PERCENTAGE, DEF_MOD_PERCENTAGE) — ExecSFTypeLandscapeCombatMod

Effect: Set attack/defense landscape modifier for an SF type.

Arguments: 1. SFType# · 2. LandscapeType# (-1=all) · 3. Off Mod in percentage points. 100=noMod, 50=half, 200=double · 4. Def Mod in percentage points. 100=noMod, 50=half, 200=double

Exe(341, SFTYPE, REINFORCEMENTTYPE) — ExecSFTypeReinforcementType

Effect: Set reinforcement-type field of a subformation type.

Arguments: 1. SFType · 2. ReinforcementType

EXECS 351–360

Execs modify scenario state or produce a defined event action.

Exe(351, LOC, STAT, MODE_0_1, VALUE) — ExecLocStats

Effect: Modify the location statistics of a location instance.

Arguments: 1. Loc# · 2. Stat # · 3. Mode 0/1=%, 2=absolute new value · 4. Value

Note: Stats: 1=supply points, 2=fuel points, 3=mode var, 4=fixed var.

Exe(353, UNIT, STAT, VALUE) — ExecChangeUnitStats

Effect: Modify an individual unit's statistics by numeric stat selector.

Arguments: 1. Unit # · 2. Stat # · 3. Value

Note: Stat # 1=Readiness, 2=Entrenchment, 3=Morale, 4=Experience.

Exe(354, X, Y, LOCTYPE, PEOPLE) — ExecChangeLocationType2

Effect: Change the secondary location type at a hex (Location2).

Arguments: 1. X · 2. Y · 3. Loctype (-1=remove any location) · 4. People #

Note: Only use this for Supply Base/Source location types.

Exe(355, X, Y, FUEL_POINTS_X, OVERRULE_RECIPIENT_REGI) — ExecGiveFuel

Effect: Distribute fuel points to locations/units using supply rules.

Arguments: 1. X · 2. Y · 3. Fuel Points on x,y · 4. Overrule recipient regime with regime #

Exe(356, MOVETYPE, TRANSPORT_TYPE, MODE) — ExecMoveTypeTransportType

Effect: Modify movement-type transport settings.

Arguments: 1. MoveType# · 2. Transport Type (0=none, 1=Transporter, 2=Attachable) · 3. Mode (0/1=Truck, 2=Train)

Exe(358, REGIME_SLOT, SETTING_1ALTERNATE_CARD, VALUE) — ExecSetRegimeSetting

Effect: Change regime-specific setting fields selected by numeric type.

Arguments: 1. Regime Slot# · 2. Setting# 1=alternate card gfx · 3. Value

Exe(359, REGIME, SPECIFIC_UNIT, MODIFIER, MODIFIER_ABSOLUTE) — ExecModifyUnitFuel

Effect: Scale and offset current unit FUEL, filtered by regime and unit ID; clamp to zero and maximum fuel reserve.

Arguments: 1. Regime (-1=none) · 2. Specific Unit (-1=none) · 3. Modifier % (100=keep same, 50=half, 200=double) · 4. Modifier absolute (0=nill, -x , +x)

Note: Increases or decreases Fuel of the unit. cannot go over max fuel reserves.

Appendix — SFType statistic selectors

Use Chk(253, SFTYPE, STAT_NUMBER) to read one of these fields, or Exe(338, SFTYPE, STAT_NUMBER, VALUE) to set it. Both commands use the same stat selector numbers.

Stat #	Statistic	Set with Exe(338)
1	Weight	Writable
2	CarryCap	Writable
3	CanDoParadrop (Boolean 0/1)	0/1 Boolean
4	EP	Writable
5	BlowBridgePts	Writable
6	BasicSupplyNeed	Writable
7	SupplyCarry	Writable
8	ReconPts	Writable
9	HidePts	Writable
10	AntiSupply	Writable
11	AntiSupplyRange	Writable
12	AntiSupplySea	Writable
13	Initiative	Writable
14	InitiativeDef	Writable
15	Attacks	Writable
16	Frontage	Writable
17	BackBench (Boolean 0/1)	0/1 Boolean
18	ArtRange	Writable
19	FavTargetTries	Writable
20	AARange	Writable
21	KillPercent	Writable
22	RetreatPercent	Writable
23	ArtSFType	Writable
24	MaxAttacked	Writable
25	MoveRedux	Writable
26	DepletingHitpointRule	Writable
27	MaxPreventPointsUsed	Writable
28	Highest PreventChance entry	Sets all PreventChance entries
29	StartCombatRound	Writable
30	EndCombatRound	Writable
31	MoveType	Writable
32	UnitGroup	Writable
33	Ratio	Writable
34	AntiStrucPts	Writable
35	directRange	Writable
36	directModFirstHex	Writable
37	directModPerHex	Writable
38	heightLevelDiff	Writable
39	targettedByRangedChance	Writable
40	scrapable	Writable
41	SupplyMaxIn	Writable
42	unreliability	Writable